S. J. P. N. TRUST'S
HIRASUGAR INSTITUTE OF TECHNOLOGY, NIDASOSHI
Accredited at 'A' Grade by NAAC
Programmes Accredited by NBA: CSE, ECE, EEE & ME.

Department of Computer Science & Engineering

Course: Design And Analysis of Algorithms (18CS42)

Module 4: Dynamic Programming, Transitive Closure, All Pairs Shortest Path

Prof. A. A. Daptardar
Asst. Prof. , Dept. of Computer Science & Engg.,
Hirasugar Institute of Technology, Nidasoshi

Module – 4
Dynamic Programming

1. General Method
2. Multistage Graphs
3. Transitive Closure – Warshall's Algorithm
4. All Pairs-Shortest Path – Floyd's Algorithm
5. Optimal Binary Search Tree
6. Knapsack Problem
7. Bellman-Ford Algorithm
8. Travelling Sales Person Problem
9. Reliability Design

General Method

- Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions
- Dynamic programming is a technique used to obtain optimal solution to given real time examples
- Examples:- Knapsack Problem, Shortest Path

Example 5.1 [Knapsack] The solution to the knapsack problem (Section 4.2) can be viewed as the result of a sequence of decisions. We have to decide the values of $x_i, 1 \leq i \leq n$. First we make a decision on x_1, then on x_2, then on x_3, and so on. An optimal sequence of decisions maximizes the objective function $\sum p_i x_i$. (It also satisfies the constraints $\sum w_i x_i \leq m$ and $0 \leq x_i \leq 1$.) □

Example 5.2 [Optimal merge patterns] This problem was discussed in Section 4.7. An optimal merge pattern tells us which pair of files should be merged at each step. As a decision sequence, the problem calls for us to decide which pair of files should be merged first, which pair second, which pair third, and so on. An optimal sequence of decisions is a least-cost sequence. □

Example 5.3 [Shortest path] One way to find a shortest path from vertex i to vertex j in a directed graph G is to decide which vertex should be the second vertex, which the third, which the fourth, and so on, until vertex j is reached. An optimal sequence of decisions is one that results in a path of least length. □

- In Dynamic Programming, an optimal sequence of decisions is obtained by making an explicit appeal to the Principle of Optimality

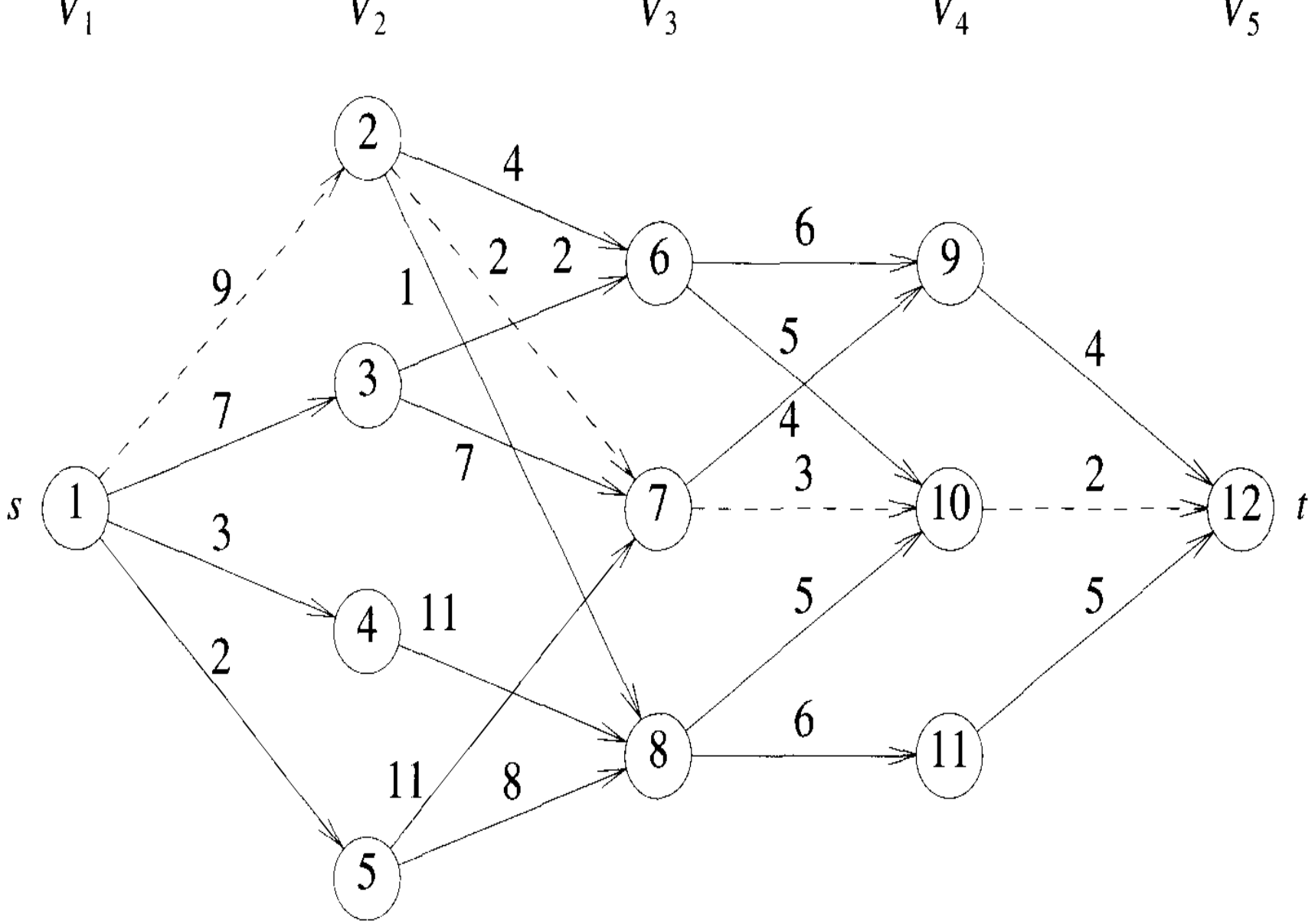
Principle of Optimality

- It states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

Multistage Graph

Multistage Graphs

- A multistage graph G=(V, E) is a directed graph in which the vertices are divided into k ≥ 2 disjoint sets V_i, 1 ≤ i ≤ k.
- In addition, if(u, v) is an edge in E, then u ϵ V_i and v ϵ V_{i+1} for some i, 1 ≤ i ≤ k.
- The sets V_1 and V_k are such that $|V_1|=|V_k|=1$.
- Let s and t, respectively, be vertices in the graph V_1 and V_k.
- Let c(i,j) be the cost of the edge <i,j> .
- The Multistage graph problem is to find a minimum cost path from source vertex s to sink vertex t in given multistage graph
- The next figure shows a five-stage graph


V_1
V_2
V_3
V_4
V_5
s
1
2
3
4
5
6
7
8
9
10
11
12
t
9
4
1
2
2
6
7
3
7
4
6
5
4
3
2
11
2
5
5
11
8
6

Formula to find Minimum Cost from source s to sink t in Multistage Graph

$$cost(i, j) = \min \{ c(j, l) + cost(i+1, l) \}$$

Where,

1. cost(i, j) is cost of vertex j in stage i
2. c(j, l) is cost of edge from vertex j to vertex l
3. cost(i+1, l) is cost of vertex l in stage i+1
4. $l \in V_{i+1}$
5. $\langle j, l \rangle \in E$

Forward Approach

```
1   Algorithm FGraph(G, k, n, p)
2   // The input is a k-stage graph G = (V, E) with n vertices
3   // indexed in order of stages. E is a set of edges and c[i, j]
4   // is the cost of ⟨i, j⟩. p[1 : k] is a minimum-cost path.
5   {
6       cost[n] := 0.0;
7       for j := n − 1 to 1 step −1 do
8       { // Compute cost[j].
9           Let r be a vertex such that ⟨j, r⟩ is an edge
10          of G and c[j, r] + cost[r] is minimum;
11          cost[j] := c[j, r] + cost[r];
12          d[j] := r;
13      }
14      // Find a minimum-cost path.
15      p[1] := 1; p[k] := n;
16      for j := 2 to k − 1 do p[j] := d[p[j − 1]];
17  }
```

Multistage-Graph Example

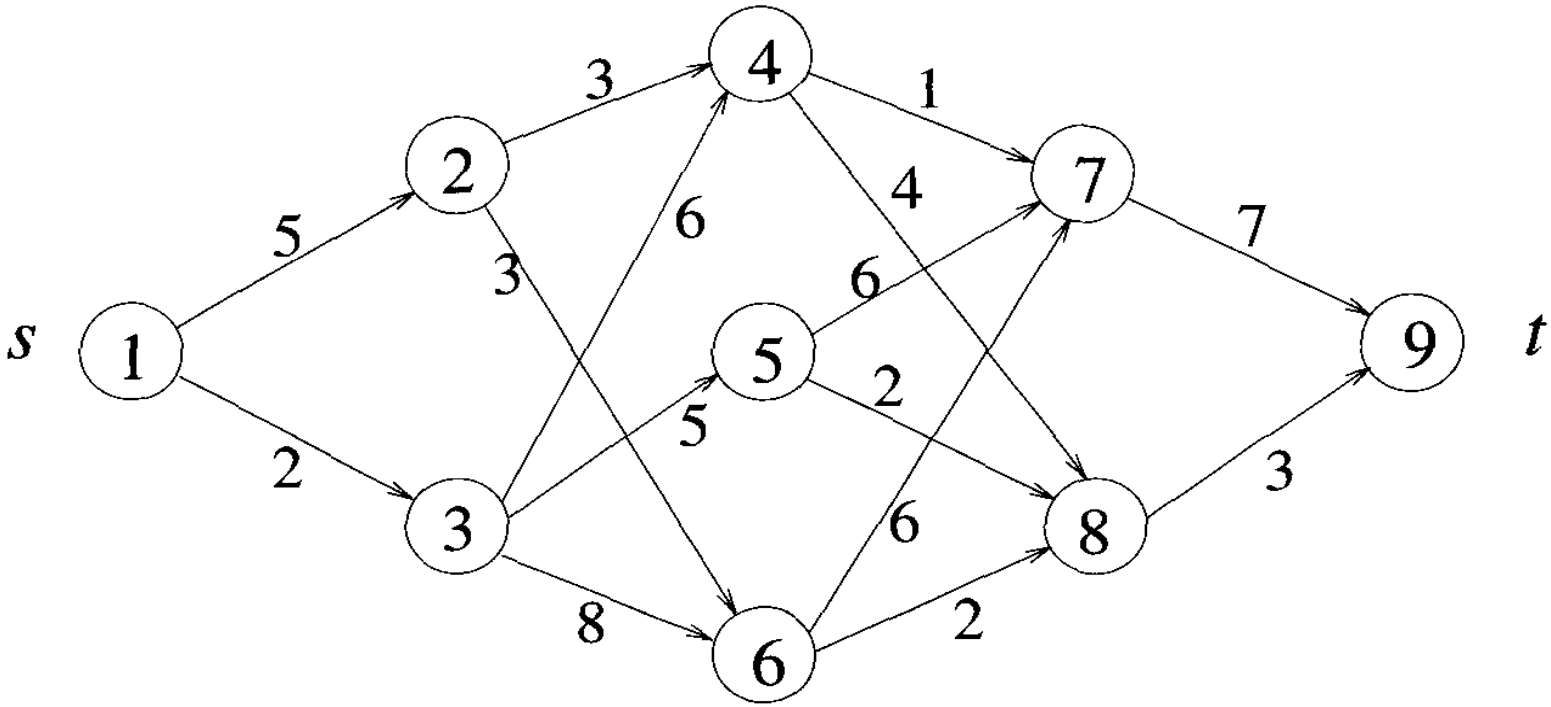
1. cost(5,12) = 0
2. cost(4,9)=4 cost(4,10)=2 cost(4,11)=5
3. cost(3,6) = min { c(6,9) + cost(4,9), c(6,10) + cost(4,10) } = min { 6+4, 5+2} = min {10, 7} = 7
4. cost(3,7) = min { c(7,9) + cost(4,9), c(7,10) + cost(4,10) } = min { 4+4, 3+2} = min {8, 5} = 5
5. cost(3,8) = min { c(8,10) + cost(4,10), c(8,11) + cost(4,11) } = min { 5+2, 6+5} = min {7, 11} = 7
6. cost(2,2) = min { c(2,6) + cost(3,6), c(2,7) + cost(3,7), c(2,8) + cost(3,8) }
   = min { 4+7, 2+5, 1+7 } = min { 11, 7, 8} = 7
7. cost(2,3) = min { c(3,6) + cost(3,6), c(3,7) + cost(3,7) } = min { 2+7, 7+5 } = min { 9, 12} = 9
8. cost(2,4) = min { c(4,8) + cost(3,8) } = min { 11+7} = min { 18 } = 18
9. cost(2,5) = min { c(5,7) + cost(3,7), c(5,8) + cost(3,8) } = min { 11+5, 8+7} = min {16, 15} = 15
10. **cost(1,1)** = min **{** **c(1,2)** + cost(2,2), **c(1,3**) + cost(2,3), c(1,4) + cost(2,4), c(1,5) + cost(2,5) }
    = min { 9+7, 7+9, 3+18, 2+15} = min { **16, 16**, 21, 17} = 16

Two optimal solutions are possible for above given example-

cost(1,1) = 2
cost(2,2) = 7
cost(3,7) = 10
cost(4,10) = 12
Edges from source to sink 1 -> 2 -> 7 ->10 -> 12
Cost from source to sink = 9 + 2 + 3 + 2
= 16

cost(1,1) =3
cost(2,3) = 6
cost(3,6) = 10
cost(4,10) = 12
Edges from source to sink 1 -> 3 -> 6 -> 10 -> 12
cost from source to sink = 7 + 2 + 5 + 2
= 16

- Find the minimum cost path from s to t in the multistage graph shown below using forward approach



cost(5,9) = 0

cost(4,7) = 7

cost(4,8) = 3

cost(3,4) = min{c(4,7)+cost(4,7), c(4,8) + cost(4,8)}

= min{ 1+7, 4+3} = min{ 8, 7} = 7

cost(3,5) = min{c(5,7)+cost(4,7), c(5,8) + cost(4,8)}

= min{ 6+7, 2+3} = min{ 13, 5} = 5

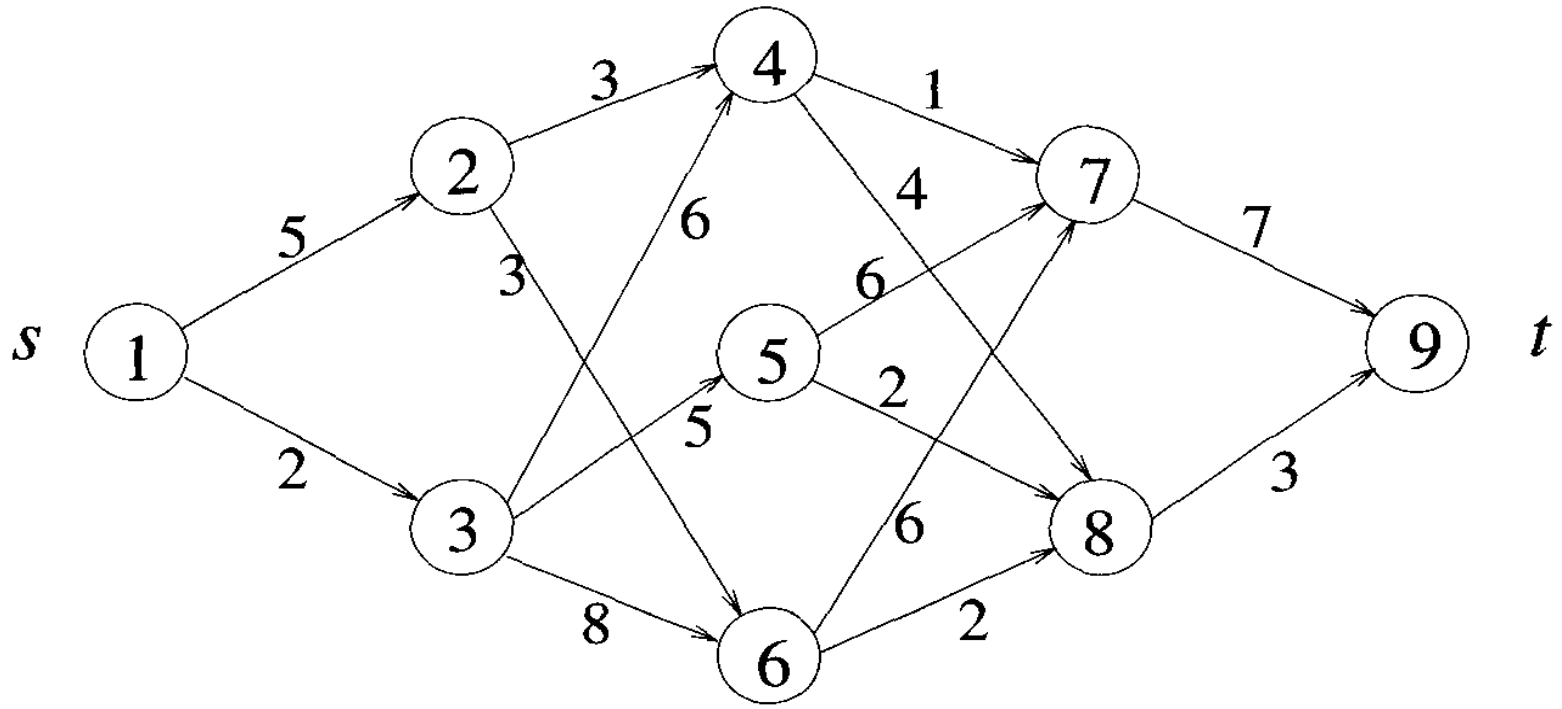
cost(3,6) = min{c(6,7)+cost(4,7), c(6,8) + cost(4,8)}

= min{ 6+7, 2+3} = min{ 13, 5} = 5

cost(2,2) = min{c(2,4)+cost(3,4), c(2,6) + cost(3,6)}
= min{ 3+7, 3+5} = min{ 10, 8} = 8
cost(2,3) = min{c(3,4)+cost(3,4), c(3,5)+cost(3,5),c(3,6) + cost(3,6)}
= min{ 6+7, 5+5,8+5} = min{ 13,10,13} = 10
cost(1,1) = min{c(1,2)+cost(2,2), c(1,3) + cost(2,3)}
= min{ 5+8, 2+10} = min{ 13, 12} = 12

One optimal solution is possible for above given example-

cost(1,1) = 3
cost(2,3) = 5
cost(3,5) = 8
cost(4,8) = 9
Edges from source to sink 1 -> 3 -> 5 -> 8 -> 9
Cost from source to sink = 2 + 5 + 2 + 3
= 12


s
1
2
3
4
5
6
7
8
9
t
5
2
3
8
3
6
5
6
4
1
6
2
7
7
3

Backward Approach

```
1   Algorithm BGraph(G, k, n, p)
2   // Same function as FGraph
3   {
4       bcost[1] := 0.0;
5       for j := 2 to n do
6       { // Compute bcost[j].
7           Let r be such that ⟨r, j⟩ is an edge of
8           G and bcost[r] + c[r, j] is minimum;
9           bcost[j] := bcost[r] + c[r, j];
10          d[j] := r;
11      }
12      // Find a minimum-cost path.
13      p[1] := 1; p[k] := n;
14      for j := k − 1 to 2 do p[j] := d[p[j + 1]];
15  }
```

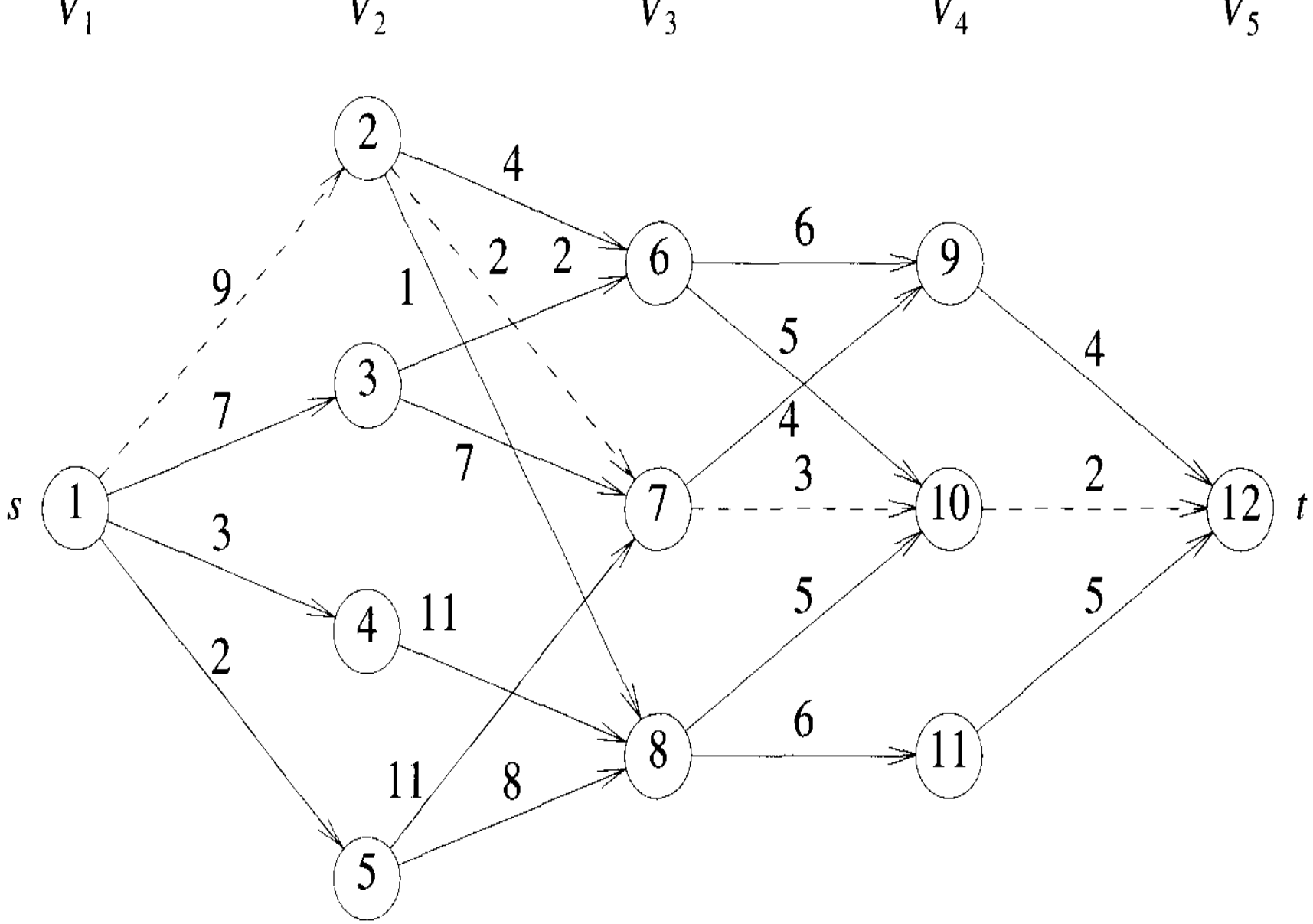
The multistage graph problem can also be solved using the backward approach. Let $bp(i, j)$ be a minimum-cost path from vertex s to a vertex j in V_i. Let $bcost(i, j)$ be the cost of $bp(i, j)$. From the backward approach we obtain

$$bcost(i, j) = \min_{\substack{l \in V_{i-1} \\ \langle l, j \rangle \in E}} \{bcost(i-1, l) + c(l, j)\} \tag{5.6}$$

Since $bcost(2, j) = c(1, j)$ if $\langle 1, j \rangle \in E$ and $bcost(2, j) = \infty$ if $\langle 1, j \rangle \notin E$, $bcost(i, j)$ can be computed using (5.6) by first computing $bcost$ for $i = 3$, then for $i = 4$, and so on. For the graph of Figure 5.2, we obtain

$$\begin{aligned}
bcost(3,6) &= \min \{bcost(2,2) + c(2,6), bcost(2,3) + c(3,6)\} \\
&= \min \{9 + 4, 7 + 2\} \\
&= 9 \\
bcost(3,7) &= 11 \\
bcost(3,8) &= 10 \\
bcost(4,9) &= 15
\end{aligned}$$

$$
\begin{aligned}
bcost(4,10) &= 14 \\
bcost(4,11) &= 16 \\
bcost(5,12) &= 16
\end{aligned}
$$


V_1
V_2
V_3
V_4
V_5
s
t
1
2
3
4
5
6
7
8
9
10
11
12
9
7
3
2
4
1
2
2
7
3
11
8
11
6
5
4
5
6
5
4
3
2
5

Multistage-Graph Example

bcost(1,1) = 0

bcost(2,2)=9

bcost(2,3)=7

bcost(2,4) = 3

bcost(2,5) = 2

bcost(3,6) = min { bcost(2,2) + c(2,6), bcost(2,3) + c(3,6)} = min { 9+4, 7+2} = min {13, 9} = 9

bcost(3,7) = min {bcost(2,2) + c(2,7), bcost(2,3) + c(3,7), bcost(2,5) + c(5,7)}

= min { 9+2, 7+7, 2+11} = min {11, 14, 13} = 11

bcost(3,8) = min {bcost(2,2) + c(2,8), bcost(2,4) + c(4,8), bcost(2,5) + c(5,8) }

= min { 9+7, 3+11, 2+8} = min {16, 14, 10} = 10

bcost(4,9) = min { bcost(3,6) + c(6,9), bcost(3,7) + c(7,9)}

= min { 9+6, 11+4} = min { 15, 15} = 15

bcost(4,10) = min { bcost(3,6) + c(6,10), bcost(3,7) + c(7,10), bcost(3,8) + c(8,10)}

= min { 9+5, 11+3, 10 + 5 } = min { 14, 14, 15} = 14

bcost(4,11) = min {bcost(3,8) + c(8,11)}

= min { 10+6} = min { 16} = 16

bcost(5,12) = min {bcost(4,9) + c(9,12), bcost(4,10) + c(10,12), bcost(4,11) + c(11,12) }

= min { 15+4, 14+2, 16 + 5} = min {19, 16, 21} = 16

Two optimal solutions are possible for above given example-

cost(1,1) = 2

cost(2,2) = 7

cost(3,7) = 10

cost(4,10) = 12

Edges from source to sink 1 -> 2 -> 7 ->10 -> 12 -> 10 -> 12

Cost from source to sink = 9 + 2 + 3 + 2 = 16

cost(1,1) =3

cost(2,3) = 6

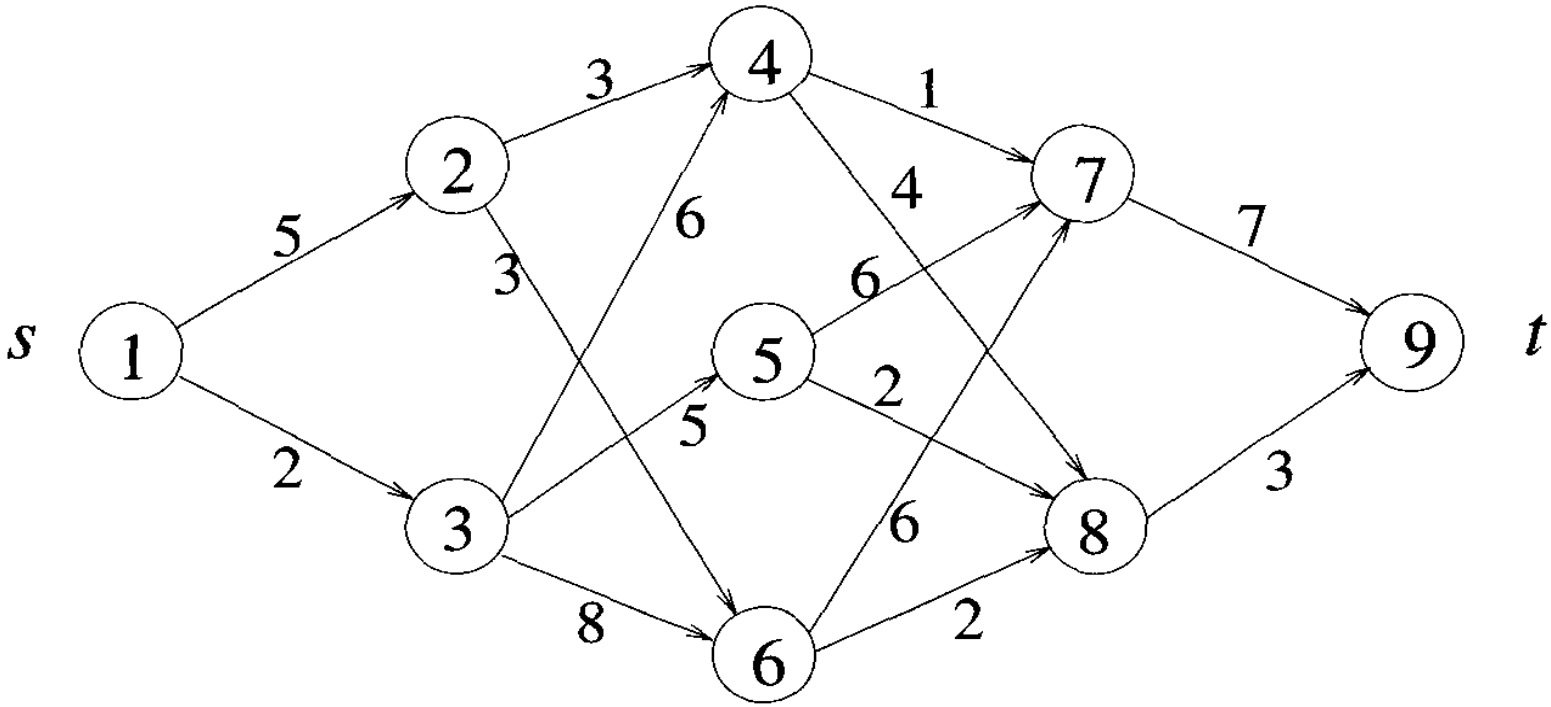
cost(3,6) = 10

cost(4,10) = 12

Edges from source to sink 1 -> 3 -> 6 ->

cost from source to sink = 7 + 2 + 5 + 2 = 16

- Find the minimum cost path from s to t in the multistage graph shown below using Backward approach



bcost(1,1) = 0

bcost(2,2) = 5

bcost(2,3) = 2

bcost(3,4) = min{ bcost(2,2)+c(2,4), bcost(2,3)+c(3,4)}

= min { 5 + 3, 2+ 6} = min {8,8} = 8

bcost(3,5) = min{ bcost(2,3)+c(3,5)}

= min { 2+ 5} = min {7} = 7

bcost(3,6) = min{ bcost(2,2)+c(2,6), bcost(2,3)+c(3,6)}

= min { 5 + 3, 2+ 8} = min {8,10} = 8

bcost(4,7) = min{ bcost(3,4)+c(4,7), bcost(3,5)+c(5,7**),** bcost(3,6)+c(6,7)**}**

= min { 8 + 1, 7+ 6, 8+ 6} = min {9,13,14} = 9

bcost(4,8) = min{ bcost(3,4)+c(4,8), bcost(3,5)+c(5,8**),** bcost(3,6)+c(6,8)**}**

= min { 8+4, 7+2 , 8+2 } = min {12,9,10} = 9

bcost(5,9) = min{ bcost(4,7)+c(7,9), bcost(4,8)+c(8,9**)}**

= min { 9 + 7, 9+ 3} = min {16,12} = 12

One optimal solution is possible for above given example-

cost(1,1) = 3

cost(2,3) = 5

cost(3,5) = 8

cost(4,8) = 9

Edges from source to sink 1 -> 3 -> 5 -> 8 -> 9

Cost from source to sink = 2 + 5 + 2 + 3

= 12

Transitive Closure
Warshall's Algorithm

Transition Closure Definition

- The Transitive closure of a directed graph with n vertices can be defined as the n-by-n Boolean matrix $T = \{t_{ij}\}$, in which the element in the i^{th} row ($1 \le i \le n$) and the j^{th} column ($1 \le j \le n$) is 1 if there exists a nontrivial directed path from the i^{th} vertex to the j^{th} vertex otherwise, t_{ij} is 0

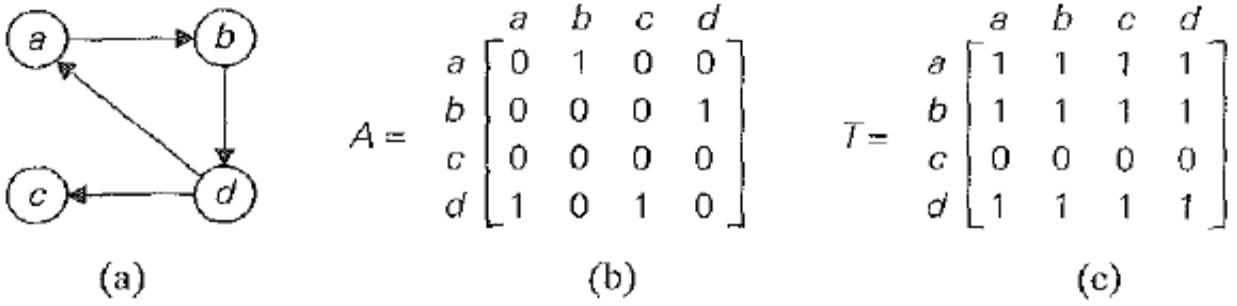


FIGURE 8.2 (a) Digraph. (b) Its adjacency matrix. (c) Its transitive closure.

Warshall's algorithm

- Warshall's algorithm constructs the transitive closure of a given digraph with n vertices through a series of n-by-n boolean matrices:

$$R^{(0)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}. \qquad (8.5)$$

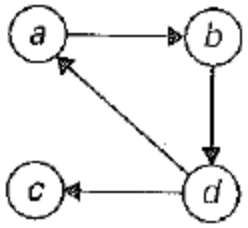
- Each of these matrices provides certain information about directed paths in the digraph.
- Specifically, the element r_{ij}^{k} in the i^{th} row and j^{th} column of matrix $R^{(k)}$ (k = 0, 1, ... , n) is equal to 1 if and only if there exists a directed path (of a positive length) from the i^{th} vertex to the j^{th} vertex with each intermediate vertex, if any, numbered not higher than k.

- we have the following formula for generating the elements of matrix $R^{(k)}$ from the elements of matrix $R^{(k-1)}$:

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \text{ or } \left(r_{ik}^{(k-1)} \text{ and } r_{kj}^{(k-1)}\right). \qquad (8.7)$$

- Formula (8.7) is at the heart of Warshall's algorithm.
- This formula implies the following rule for generating elements of matrix $R^{(k)}$ from elements of matrix $R^{(k-1)}$, which is particularly convenient for applying Warshall's algorithm by hand:
  - If an element r_{ij} is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$
  - If an element r_{ij} is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if the element in its row i and column k and the element in its column j and row k are both 1's in $R^{(k-1)}$

- Thus, the series starts with $R^{(0)}$ which does not allow any intermediate vertices in its paths; hence, $R^{(0)}$ is nothing else but the adjacency matrix of the digraph.
- $R^{(1)}$ contains the information about paths that can use the first vertex as intermediate; thus, it may contain more ones than $R^{(0)}$.
- In general, each subsequent matrix in series (8.5) has one more vertex to use as intermediate for its paths than its predecessor and hence may, but does not have to, contain more ones.
- The last matrix in the series, $R^{(n)}$, reflects paths that can use all *n vertices of* the digraph as intermediate and hence is nothing else but the digraph's transitive **closure.**



$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

Through vertex a

(d, a) = 1 & (a, b) = 1, Hence (d, b) = 1

$R^{(0)}$ Results into $R^{(1)}$

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

Through vertex b

(a, b) = 1 & (b, d) = 1, Hence (a, d) = 1

(d, b) = 1 & (b, d) = 1, Hence (d, d) = 1

$R^{(1)}$ Results into $R^{(2)}$

$R^{(2)} =$

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	**1**	**1**

Through vertex c

(d, c) = 1 but no path from c to others. Hence, $R^{(3)}$ remains same as $R^{(2)}$

$R^{(3)} =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

Through vertex d

(a, d) = 1 & (d, a) = 1, Hence (a, a) = 1

(a, d) = 1 & (d, b) = 1, Hence (a, b) = 1

(a, d) = 1 & (d, c) = 1, Hence (a, c) = 1

(a, d) = 1 & (d, d) = 1, Hence (a, d) = 1

$R^{(4)} =$

	a	b	c	d
a	**1**	1	**1**	1
b	**1**	**1**	**1**	1
c	0	0	0	0
d	1	1	1	1

(b, d) = 1 & (d, a) = 1, Hence (b, a) = 1

(b, d) = 1 & (d, b) = 1, Hence (b, b) = 1

(b, d) = 1 & (d, c) = 1, Hence (b, c) = 1

(b, d) = 1 & (d, d) = 1, Hence (b, d) = 1

$R^{(3)}$ Results into $R^{(4)}$

Finally $R^{(4)}$ is Transitive Closure of above given Diagraph

Warshall's Algorithm

ALGORITHM *Warshall*$(A[1..n, 1..n])$

//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
$R^{(0)} \leftarrow A$
for $k \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $(R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j])$
return $R^{(n)}$

Observations

- We can speed up the above implementation of Warshall's algorithm for some inputs by restructuring its innermost loop.
- Another way to make the algorithm run faster is to treat matrix rows as bit strings and employ the bitwise or operation available in most modern computer languages.

- Apply Warshall's algorithm to find the transitive closure of the digraph defined by the following adjacency matrix.

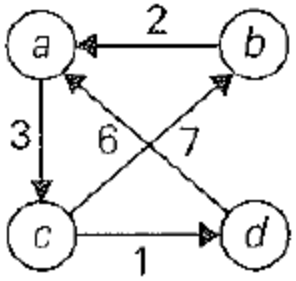
$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Floyd’s Algorithm

Introduction

- Given a weighted connected graph (undirected or directed), the all-pairs shortest paths problem asks to find the distances (the lengths of the shortest paths) from each vertex to all other vertices.
- It is convenient to record the lengths of shortest paths in an n-by-n matrix D called the distance matrix: the element d_{ij} in the i^{th} row and the j^{th} column of this matrix indicates the length of the shortest path from the i^{th} vertex to the j^{th} vertex ($1 \le i, j \le n$).

- We can generate the distance matrix with an algorithm that is very similar to Warshall's algorithm. It is called Floyd's algorithm, after its inventor R. Floyd [Flo62].
- It is applicable to both undirected and directed weighted graphs provided that they do not contain a cycle of a negative length.



(a)

$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

(b)

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

(c)

FIGURE 8.5 (a) Digraph. (b) Its weight matrix. (c) Its distance matrix.

- Floyd's algorithm computes the distance matrix of a weighted graph with n vertices through a series of n-by-n matrices:

$$D^{(0)}, \ldots, D^{(k-1)}, D^{(k)}, \ldots, D^{(n)}. \qquad (8,8)$$

- Each of these matrices contains the lengths of shortest paths with certain constraints on the paths considered for the matrix in question.
- Specifically, the element d_{ij}^{k} in the i^{th} row and the j^{th} column of matrix n<k) (k = 0, 1, ... , n) is equal to the length of the shortest path among all paths from the i^{th} vertex to the j^{th} vertex with each intermediate vertex, if any, numbered not higher thank.
- In particular, the series starts with $D^{(0)}$, which does not allow any intermediate vertices in its paths; hence, $D^{(0)}$ is nothing but the weight matrix of the graph.
- The last matrix in the series, $D^{(n)}$, contains the lengths of the shortest paths among all paths that can use all n vertices as intermediate and hence is nothing but the distance matrix being sought.

- We can compute all the elements of each matrix $D^{(k)}$ from its immediate predecessor $D^{(k-1)}$in series (8.8).
- Let $d_{ij}^{(k)}$ be the element in the i^{th} row and the j^{th} column of matrix $D^{(k)}$. This means that $d_{ij}^{(k)}$ is equal to the length of the shortest path among all paths from the i^{th} vertex vi to the j^{th} vertex vj with their intermediate vertices numbered not higher than k:
- Vi, a list of intermediate vertices each numbered not higher than k, vj . (8.9)
- Taking into account the lengths of the shortest paths in both subsets leads to the following recurrence:

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \quad \text{for } k \geq 1, \quad d_{ij}^{(0)} = w_{ij}. \qquad \textbf{(8.10)}$$

- To put it another way, the element in the i^{th} row and the j^{th} column of the current distance matrix $D^{(k-1)}$ is replaced by the sum of the elements in the same row i and the k^{th} column and in the same column j and the k^{th} column if and only if the latter sum is smaller than its current value.

Floyd's Algorithm

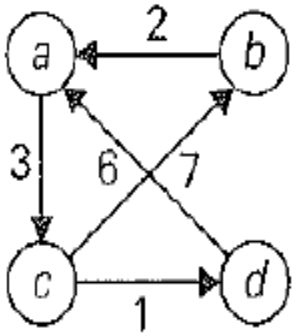
(All Pairs shortest path problem)

to find distance from each vertex to all other vertices

ALGORITHM *Floyd*$(W[1..n, 1..n])$

//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
$D \leftarrow W$ //is not necessary if W can be overwritten
for $k \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $D[i, j] \leftarrow \min\{D[i, j], D[i, k] + D[k, j]\}$
return D

Floyd's Algorithm Example



(a)

$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

(b)

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

(c)

FIGURE 8.5 (a) Digraph. (b) Its weight matrix. (c) Its distance matrix.

Floyd's Algorithm Example

$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

$D^{(3)} =$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	**6**	**16**	9	0

Through vertex a

(b, a) = 2 and (a, c) = 3 (b, c) = min { (b, c), (b, a) + (a, c) }
= min { ∞, 2+3} = 5, **(b, c) = 5**

(d, a) = 6 and (a, c) = 3 (d, c) = min { (d, c), (d, a) + (a, c) }
= min { ∞, 6+3} = 9, **(d, c) = 9**

$D^{(0)}$ Results into $D^{(1)}$

Through Vertex b

(c, b) = 7 and (b, a) = 2 (c, a) = min { (c, a), (c, b) + (b, a) }
= min {∞ , 7+2} = 9, **(c, a) = 9**

(c, b) = 7 and (b, c) = 5 (c, c) = min { (c, c), (c, b) + (b, c) }
= min { 0, 7+5} = 0, **(c, c) = 0**

$D^{(1)}$ Results into $D^{(2)}$

Through Vertex c

(a, c) = 3 & (c, a) = 9 hence, (a, a) = min { 0, 3+9 } = **(a, a)= 0**
(a, c) = 3 & (c, b) = 7 hence, (a,b) = min { ∞, 3+7 } = **(a,b)= 10**
(a, c) = 3 & (c, d) = 1 hence, (a, d) = min { ∞, 3+1 } = **(a, d)=4**

(b, c) = 5 & (c, a) = 9 hence, (b, a) = min { 2, 5+9 } = **(b, a)= 2**
(b, c) = 5 & (c, b) = 7 hence, (b, b) = min { 0, 5+7 } = **(b, b)= 0**
(b, c) = 5 & (c, d) = 1 hence, (b, d) = min { ∞, 5+1 } = **(b, d)=6**

(d, c) = 9 & (c, a) = 9 hence, (d, a) = min { 6, 9+9 } = **(d, a)= 6**
(d, c) = 9 & (c, b) = 7 hence, (d,b) = min { ∞, 9+7 } = **(d,b)= 16**
(d, c) = 9 & (c, d) = 1 hence, (d, d) = min { 0, 9+1 } = **(d, d)=0**

$D^{(2)}$ Results into $D^{(3)}$

Floyd's Algorithm Example

$$D^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Through Vertex d

(a, d) = 4 & (d, a) = 6 hence, (a, a) = min { 0, 4 + 6 } = **(a, a)= 0**

(a, d) = 4 & (d,b)=16 hence,(a,b)=min{ 10, 4+16 } = **(a,b)= 10**

(a, d) = 4 & (d, c) = 9 hence, (a, c) = min { 3, 4 + 9 } = **(a, c) = 3**

(b, d) = 6 & (d, a) = 6 hence, (b, a) = min { 2, 6+6 } = **(b, a)= 2**

(b, d) = 6 & (d, b)=16 hence,(b, b)=min { 0, 6+16 } = **(b, b)= 0**

(b, d) = 6 & (d, c) = 9 hence, (b, c) = min { 5, 6+9 } = **(b, c)=5**

(c, d) = 1 & (d, a) = 6 hence, (c, a) = min { 9, 1+6 } = **(c, a)= 7**

(c, d) = 1 & (d, b)=16 hence, (c, b) = min { 7, 1+16 } = **(c, b)= 7**

(c, d) = 1 & (d, c) = 9 hence, (c, c) = min { 0, 1+9 } = **(c, c)=0**

$D^{(3)}$ Results into $D^{(4)}$

$D^{(4)}$ is Resultant Distance Matrix for all pair shortest path

- Solve the all-pairs shortest-path problem for the digraph with the weight **matrix**

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Through vertex a

(b,a)=6 and (a,b) = 2 hence (b,b) = min{0,6+2} = **(b,b)=0**

(b,a)=6 and (a,d) = 1 hence (b,d) = min{2, 6+1} **=(b,d)=2**

(b,a) = 6 and (a,e) = 8 hence (b,e)= min{∞, 6+8}=**(b,e)=14**

(e,a)=3 and (a,b)=2 hence (e,b) = min{∞, 3+2}=**(e,b) =5**

(e,a)=3 and (a,d)=1 hence (e,d) = min{∞, 3+1}=**(e,d) =4**

(e,a)=3 and (a,e)=8 hence (e,e) = min{0, 3+8}=**(e,e) =0**

$D^{(0)}$ results in $D^{(1)}$

Through vertex b

(a,b) = 2 and (b,a) = 6 hence (a,a)=min{0,2+6}=**(a,a)=0**

(a,b) = 2 and (b,c) = 3 hence (a,c)=min{∞,2+3}=**(a,c)=5**

(a,b) = 2 and (b,d) = 3 hence (a,d)=min{1,2+2}=**(a,d)=1**

(a,b) = 2 and (b,e) = 14 hence (a,e)=min{8,2+14}=**(a,e)=8**

(e,b) = 5 and (b,a) = 6 hence (e,a)=min{3,5+6}=**(e,a)=3**

(e,b) = 5 and (b,c) = 3 hence (e,c)=min{∞,5+3}=**(e,c)=8**

(e,b) = 5 and (b,d) = 2 hence (e,d)=min{4,5+2}=**(e,d)=4**

(e,b) = 5 and (b,e) = 14 hence (e,e)=min{0,5+14}=**(e,e)=0**

$D^{(1)}$ results in $D^{(2)}$

Through the vertex c

(a,c)=5 and (c,d)=4 hence (a,d)=min{1,5+4} = **(a,d)=1**

(b,c)=3 and (c,d)=4 hence (b,d)=min{2,3+4} = **(b,d)=2**

(d,c)=2 and (c,d)=4 hence (d,d)=min{0,2+4} = **(d,d)=0**

(e,c)=8 and (c,d)=4 hence (e,d)=min{4,8+4} = **(e,d)=4**

$D^{(2)}$ results in $D^{(3)}$

Through vertex d

(a,d)=1 and (d,c)=2 hence (a,c)=min{5,1+2} = **(a,c)=3**

(a,d)=1 and (d,e)=3 hence (a,e)=min{8,1+3} = **(a,e)=4**

(b,d)=2 and (d,c)=2 hence (b,c)=min{3,2+2} = **(b,c)=3**

(b,d)=2 and (d,e)=3 hence (b,e)=min{4,2+3} = **(b,e)=4**

(c,d)=4 and (d,c)=2 hence (c,c)=min{0,4+2} = **(c,c)=0**

(c,d)=4 and (d,e)=3 hence (c,e)=min{∞,4+3} = **(c,e)=7**

0/1 Knapsack Problem

0/1 Knapsack Problem

- For given n items/objects of known weights w_1, w_2,…..w_n and values v_1, v_2,…..v_n and a knapsack of capacity W.
- Find the most valuable subset of the items that fit into the knapsack.
- We assume here that all the weights and the knapsack capacity are positive integers
- Also all items are non-divisible i.e. consider full item(1) or not consider(0)

- To design a dynamic programming algorithm, we need to derive a recurrence relation that expresses a solution to an instance of the knapsack problem in terms of solutions to its smaller subinstances.
- Let us consider an instance defined by the first i items, $1 \le i \le n$, with weights w_1, ... , w_i, values v_1, ... , v_i, and knapsack capacity j, $1 \le j \le n$.
- Let V[i, j] be the value of an optimal solution to this instance, i.e., the value of the most valuable subset of the first i items that fit into the knapsack of capacity j.
- We can divide all the subsets of the first i items that fit the knapsack of capacity j into two categories: those that do not include the i^{th} item and those that do.
- Note the following:
  - Among the subsets that do not include the i^{th} item, the value of an optimal subset is, by definition, V[i - 1, j].
  - Among the subsets that do include the i^{th} item (hence, $j - w_i \ge 0$), an optimal subset is made up of this item and an optimal subset of the first i - 1 items that fit into the knapsack of capacity $j - w_i$. The value of such an optimal subset is $v_i + V[i - 1, j - w_i]$.
- Recurrence formula for knapsack problem is

$$V[i,j] = \begin{cases} \max \{V[i-1,j], vi + V[i-1, j-wi]\} & \text{if } j-w_i \ge 0 \\ V[i-1,j] & \text{if } j-w_i \le 0 \end{cases}$$

And the Initial Conditions are-

$$V[0,j] = 0 \text{ for } j \ge 0 \text{ and } V[i,0] = 0 \text{ for } i \ge 0$$

0/1 Knapsack with Size W=5

		capacity j					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1, v_2 = 10$	2	0	10	12	22	22	22
$w_3 = 3, v_3 = 20$	3	0	10	12	22	30	32
$w_4 = 2, v_4 = 15$	4	0	10	15	25	30	37

Item	Weight	value
1	2	12
2	1	10
3	3	20
4	2	15

$V[i, j] = \max \{V[i-1, j], v_i + V[i-1, j-w_i]\}$ if $j-w_i \geq 0$

$V[1,1] = V[1-1,1]=V[0,1]=0$ as $j-w_i \leq 0$ $(1-2 \leq 0)$

$V[1,2] = \max \{ V[1-1,2], v_1+V[1-1,2-2] \} = \max \{ V[0,2], v_1+V[0,0] \} = \max \{ 0, 12+0\} = \max \{0, 12\} = 12$

$V[1,3] = \max \{ V[1-1,3], v_1+V[1-1,3-2] \} = \max \{ V[0,3], v_1+V[0,1] \} = \max \{ 0, 12+0\} = \max \{0, 12\} = 12$

$V[1,4] = \max \{ V[1-1,4], v_1+V[1-1,4-2] \} = \max \{ V[0,4], v_1+V[0,2] \} = \max \{ 0, 12+0\} = \max \{0, 12\} = 12$

$V[1,5] = \max \{ V[1-1,5], v_1+V[1-1,5-2] \} = \max \{ V[0,5], v_1+V[0,3] \} = \max \{ 0, 12+0\} = \max \{0, 12\} = 12$

$V[2,1] = \max \{ V[2-1,1], v_2+V[2-1,1-1] \} = \max \{ V[1,1], v_2+V[1,0] \} = \max \{ 0, 10+0\} = \max \{0, 10\} = 10$

$V[2,2] = \max \{ V[2-1,2], v_2+V[2-1,2-1] \} = \max \{ V[1,2], v_2+V[1,1] \} = \max \{ 12, 10+0\} = \max \{12, 10\} = 12$

$V[2,3] = \max \{ V[2-1,3], v_2+V[2-1,3-1] \} = \max \{ V[1,3], v_2+V[1,2] \} = \max \{ 12, 10+12\} = \max \{12, 22\} = 22$

$V[2,4] = \max \{ V[2-1,4], v_2+V[2-1,4-1] \} = \max \{ V[1,4], v_2+V[1,3] \} = \max \{ 12, 10+12\} = \max \{12, 22\} = 22$

$V[2,5] = \max \{ V[2-1,5], v_2+V[2-1,5-1] \} = \max \{ V[1,5], v_2+V[1,4] \} = \max \{ 12, 10+12\} = \max \{12, 22\} = 22$

V[3,1] = V[3-1,1]=V[2,1]=10

V[3,2] = V[3-1,2]=V[2,2]=12

V[3,3] = max{V[3-1,3], v_3+V[3-1,3-3]} = max { V[2,3], 20+V[2,0] } = max { 22, 20+0 } = max {22, 20} = 22

V[3,4] = max{V[3-1,4], v_3+V[3-1,4-3]} = max { V[2,4], 20+V[2,1] } = max { 22, 20+10 } = max {22, 30}=30

V[3,5] = max{V[3-1,5], v_3+V[3-1,5-3]} = max { V[2,5], 20+V[2,2] } = max { 22, 20+12 } = max {22, 32}=32

V[4,1] = V[4-1,1]=V[3,1]=10

V[4,2] = max {V[4-1,2], v_4+V[4-1,2-2]} = max {V[3,2], 15+V[3,0]} = max { 12, 15+0 }=max {12, 15} = 15

V[4,3] = max{V[4-1,3], v_4+V[4-1,3-2]} = max {V[3,3], 15+V[3,1]} = max { 22, 15+10 }=max {22, 25} = 25

V[4,4] = max {V[4-1,4], v_4+V[4-1,4-2]} = max { V[3,4], 15+V[3,2] } = max { 30, 15+12 }=max {30, 27} = 30

V[4,5] = max {V[4-1,5], v_4+V[4-1,5-2]} = max { V[3,5], 15+V[3,3] } = max { 32, 15+22 }=max {32, 37} = 37

Finally subset of items = {1, 2, 3, 4} = { 1, 1, 0, 1}

Weight of Knapsack = {1, 2, 3, 4} = {2, 1, 0, 2} = 2 + 1 + 0 + 2 = 5 also Maximum Knapsack capacity is, W=5

Maximum Profit Value = {1, 2, 3, 4} = {12, 10, 0, 15} = 12+10+0+15 = **37**

- Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

item	weight	value
1	3	\$25
2	2	\$20
3	1	\$15
4	4	\$40
5	5	\$50

capacity $W = 6$.

$$V[1,1] = V[1\text{-}1,1] = V[0,1] = 0$$

$$V[1,2] = V[1\text{-}1,2] = V[0,2] = 0$$

$$V[1,3] = \max\{ V[1\text{-}1,3], v_1+V[1\text{-}1,3\text{-}3]\} = \max\{V[0,3], v_1+V[0,0]\} = \max\{0, 25+0\} = \max\{0,25\} = 25$$

$$V[1,4] = \max\{ V[1\text{-}1,4], v_1+V[1\text{-}1,4\text{-}3]\} = \max\{V[0,4], v_1+V[0,1]\} = \max\{0, 25+0\} = \max\{0,25\} = 25$$

$$V[1,5] = \max\{ V[1\text{-}1,5], v_1+V[1\text{-}1,5\text{-}3]\} = \max\{V[0,5], v_1+V[0,2]\} = \max\{0, 25+0\} = \max\{0,25\} = 25$$

$$V[1,6] = \max\{ V[1\text{-}1,6], v_1+V[1\text{-}1,6\text{-}3]\} = \max\{V[0,6], v_1+V[0,3]\} = \max\{0, 25+0\} = \max\{0,25\} = 25$$

$$V[2,1] = V[2\text{-}1,1] = V[1,1] = 0$$

$$V[2,2] = \max\{ V[2\text{-}1,2], v_2+V[2\text{-}1,2\text{-}2]\} = \max\{V[1,2], v_2+V[1,0]\} = \max\{0, 20+0\} = \max\{0,20\} = 20$$

$$V[2,3] = \max\{ V[2\text{-}1,3], v_2+V[2\text{-}1,3\text{-}2]\} = \max\{V[1,3], v_2+V[1,1]\} = \max\{25, 20+0\} = \max\{25,20\} = 25$$

$$V[2,4] = \max\{ V[2\text{-}1,4], v_2+V[2\text{-}1,4\text{-}2]\} = \max\{V[1,4], v_2+V[1,2]\} = \max\{25, 20+0\} = \max\{25,20\} = 25$$

$$V[2,5] = \max\{ V[2\text{-}1,5], v_2+V[2\text{-}1,5\text{-}2]\} = \max\{V[1,5], v_2+V[1,3]\} = \max\{25, 20+25\} = \max\{25,45\} = 45$$

$$V[2,6] = \max\{ V[2\text{-}1,6], v_2+V[2\text{-}1,6\text{-}2]\} = \max\{V[1,6], v_2+V[1,4]\} = \max\{25, 20+25\} = \max\{25,45\} = 45$$

V[3,1] = max{ V[3-1,1], v_3+V[3-1,1-1]} = max{V[2,1], v_3+V[2,0]} = max{0, 15+0} = max{0,15} = 15
V[3,2] = max{ V[3-1,2], v_3+V[3-1,2-1]} = max{V[2,2], v_3+V[2,1]} = max{20, 15+0} = max{20,15} = 20
V[3,3] = max{ V[3-1,3], v_3+V[3-1,3-1]} = max{V[2,3], v_3+V[2,2]} = max{25, 15+20} = max{25,35} = 35
V[3,4] = max{ V[3-1,4], v_3+V[3-1,4-1]} = max{V[2,4], v_3+V[2,3]} = max{25, 15+25} = max{25,40} = 40
V[3,5] = max{ V[3-1,5], v_3+V[3-1,5-1]} = max{V[2,5], v_3+V[2,4]} = max{45, 15+25} = max{45,40} = 45
V[3,6] = max{ V[3-1,6], v_3+V[3-1,6-1]} = max{V[2,6], v_3+V[2,5]} = max{45, 15+45} = max{25,60} = 60

V[4,1] = V[4-1,1] = V[3,1] = 15
V[4,2] = V[4-1,2] = V[3,2] = 20
V[4,3] = V[4-1,3] = V[3,3] = 35
V[4,4] = max{ V[4-1,4], v_4+V[4-1,4-4]} = max{V[3,4], v_4+V[3,0]} = max{40, 40+0} = max{40,40} = 40
V[4,5] = max{ V[4-1,5], v_4+V[4-1,5-4]} = max{V[3,5], v_4+V[3,1]} = max{45, 40+15} = max{45,55} = 55
V[4,6] = max{ V[4-1,6], v_4+V[4-1,6-4]} = max{V[3,6], v_4+V[3,2]} = max{60, 40+20} = max{60,60} = 60

V[5,1] = V[5-1,1] = V[4,1] = 15
V[5,2] = V[5-1,2] = V[4,2] = 20
V[5,3] = V[5-1,3] = V[4,3] = 35
V[5,4] = V[5-1,4] = V[4,4] = 40
V[5,5] = max{ V[5-1,5], v_5+V[5-1,5-5]} = max{V[4,5], v_5+V[4,0]} = max{55, 50+0} = max{55,50} = 55
V[5,6] = max{ V[5-1,6], v_5+V[5-1,6-5]} = max{V[4,6], v_5+V[4,1]} = max{60, 50+15} = max{60,65} = 65

Finally subset of items = {1, 2, 3, 4,5} = { 0, 0, 1, 0, 1}

Weight of Knapsack = {1, 2, 3, 4, 5} = {0, 0, 1, 0, 5} = 0 + 0 + 1 + 0 + 5 = 6 also Maximum Knapsack capacity is, W=6

Maximum Profit Value = {1, 2, 3, 4, 5} = {0, 0, 15, 0, 50} = 0+0+15+0+50 = **65**

Single Source Shortest Path
Bellman-Ford Algorithm

Bellman-Ford Algorithm

- Single source shortest path is a problem in which consider one source vertex in a given weighted connected graph and find shortest paths to all its other vertices from source vertex
- Dijkstra's algorithm do not find the optimal path if graph having negative edges
- When negative edge lengths are permitted, we require that the graph have no cycles of negative lengths.
- When there are no cycles of negative length, there is a shortest path between any two vertices of an n-vertex graph that has atmost n-1 edges on it.

Let $dist^{\ell}[u]$ be the length of a shortest path from the source vertex v to vertex u under the constraint that the shortest path contains at most ℓ edges. Then, $dist^{1}[u] = cost[v, u]$, $1 \leq u \leq n$. As noted earlier, when there are no cycles of negative length, we can limit our search for shortest paths to paths with at most $n - 1$ edges. Hence, $dist^{n-1}[u]$ is the length of an unrestricted shortest path from v to u.

Our goal then is to compute $dist^{n-1}[u]$ for all u. This can be done using the dynamic programming methodology. First, we make the following observations:

1. If the shortest path from v to u with at most k, $k > 1$, edges has no more than $k - 1$ edges, then $dist^{k}[u] = dist^{k-1}[u]$.

2. If the shortest path from v to u with at most k, $k > 1$, edges has exactly k edges, then it is made up of a shortest path from v to some vertex j followed by the edge $\langle j, u \rangle$. The path from v to j has $k - 1$ edges, and its length is $dist^{k-1}[j]$. All vertices i such that the edge $\langle i, u \rangle$ is in the graph are candidates for j. Since we are interested in a shortest path, the i that minimizes $dist^{k-1}[i] + cost[i, u]$ is the correct value for j.

These observations results in the following Recurrence formula of Bellman-Ford algorithm is

$$\mathbf{dist^k[u] = min\ \{\ dist^{k-1}[u],\ min_i\ \{\ dist^{k-1}[i] + cost[i, u]\}\}}$$

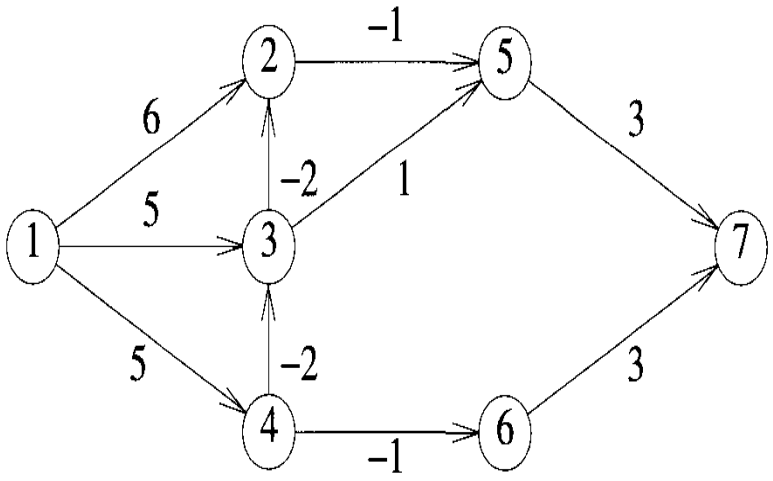
Where,

- **$dist^k[u]$** is length of shortest path from source to vertex u
- **k** are iterations in Bellman-ford algorithm for k = 2, 3, ……,n-1
- **i** are the individual vertices in given graph G
- **n** is the total number of vertices in given graph G

Bellman-Ford Algorithm

```
1   Algorithm BellmanFord(v, cost, dist, n)
2   // Single-source/all-destinations shortest
3   // paths with negative edge costs
4   {
5       for i := 1 to n do // Initialize dist.
6           dist[i] := cost[v, i];
7       for k := 2 to n − 1 do
8           for each u such that u ≠ v and u has
9                   at least one incoming edge do
10              for each ⟨i, u⟩ in the graph do
11                  if dist[u] > dist[i] + cost[i, u] then
12                      dist[u] := dist[i] + cost[i, u];
13  }
```

- Find the shortest paths from the node 1 to every other node in the graph given below using the Bellman and Ford Algorithm.



(a) A directed graph

Formula - $dist^k[u] = min \{ dist^{k-1}[u], min_i \{ dist^{k-1}[i] + cost[i, u]\}\}$

When k=1 then

$dist^1[1]=0$, $dist^1[2]=6$, $dist^1[3]=5$, $dist^1[4]=5$, $dist^1[5]= \infty$, $dist^1[6]=\infty$, $dist^1[7]=\infty$

When k=2 then

$dist^2[2] = min \{ dist^1[2], min_i \{dist^1[i] + cost[i,2] \}\}$ (where i= 1 to 7 except 2

$= min \{dist^1[2], min \{dist^1[1] + cost[1,2], dist^1[3] + cost[3,2], dist^1[4] + cost[4,2], dist^1[5] + cost[5,2], dist^1[6] + cost[6,2], dist^1[7] + cost[7,2]\}\}$

$= min \{ 6, min\{0+6, 5+(-2), 5+\infty, \infty+\infty, \infty+\infty, \infty+\infty\}\}$

$= min \{ 6, min\{6, 3, \infty, \infty, \infty, \infty\}\} = min\{6,3\} =$ **3**

$dist^2[3] = min \{ dist^1[3], min_i\ dist^1[i] + cost[i,3] \}$ (where i= 1 to 7 except 3)

$= min \{dist^1[3], min \{dist^1[1] + cost[1,3], dist^1[2] + cost[2,3], dist^1[4] + cost[4,3], dist^1[5] + cost[5,3], dist^1[6] + cost[6,3], dist^1[7] + cost[7,3]\}\}$

$= min \{ 5, min\{0+5, 6+\infty, 5+(-2), \infty+\infty, \infty+\infty, \infty+\infty\}\}$

$= min \{ 5, min\{5, \infty, 3, \infty, \infty, \infty \}\} = min\{5,3\} =$ **3**

$dist^2[4] = min \{ dist^1[4], min_i\ dist^1[i] + cost[i,3] \}$ (where i= 1 to 7 except 4)

$= min \{dist^1[4], min \{dist^1[1] + cost[1,4], dist^1[2] + cost[2,4], dist^1[3] + cost[3,4], dist^1[5] + cost[5,4], dist^1[6] + cost[6,4], dist^1[7] + cost[7,4]\}\}$

$= min \{ 5, min\{0+5, 6+\infty, 5+ \infty, \infty+\infty, \infty+\infty, \infty+\infty\}\}$

$= min \{ 5, min\{5, \infty, \infty, \infty, \infty, \infty \}\} = min\{5,5\} =$ **5**

Similarly when k=6

$dist^2[5] = \min \{ dist^1[5], \min_i dist^1[i] + cost[i,5] \}$ (where i= 1 to 7 except 5)

$= \min \{dist^1[5], \min \{dist^1[1] + cost[1,5], dist^1[2] + cost[2,5], dist^1[3] + cost[3,5], dist^1[4] + cost[4,5],$
$dist^1[6] + cost[6,5], dist^1[7] + cost[7,5]\}\}$

$= \min \{\infty, \min\{0+ \infty, 6+(-1), 5+1, 5+\infty, \infty+\infty, \infty+\infty\}\}$

$= \min \{\infty , \min\{ \infty, 5, 6, \infty, \infty, \infty \}\} = \min\{\infty,5\} =$ **5**

$dist^2[6] = \min \{ dist^1[6], \min_i dist^1[i] + cost[i,6] \}$ (where i= 1 to 7 except 6)

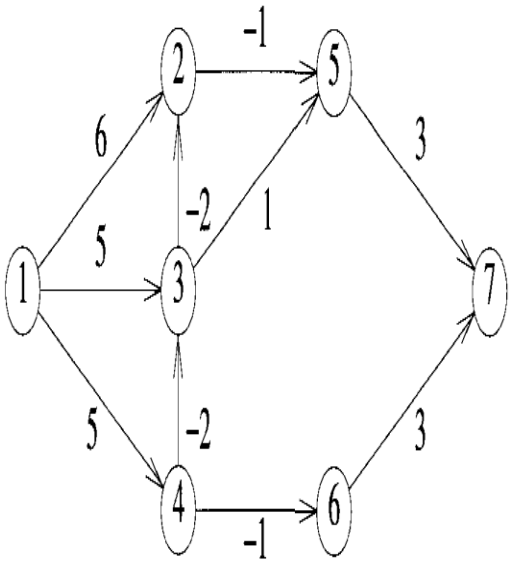
$= \min \{dist^1[6], \min \{dist^1[1] + cost[1,6], dist^1[2] + cost[2,6], dist^1[3] + cost[3,6], dist^1[4] + cost[4,6],$
$dist^1[5] + cost[5,6], dist^1[7] + cost[7,6]\}\}$

$= \min \{\infty, \min\{0+ \infty, 6+ \infty, 5+ \infty, 5+(-1), \infty+\infty, \infty+\infty\}\}$

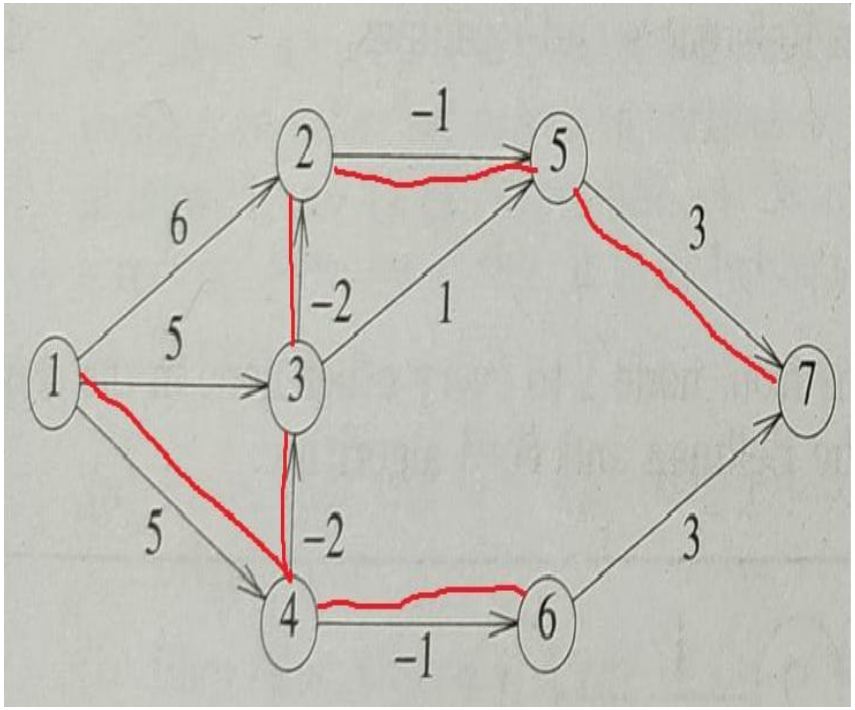
$= \min \{\infty , \min\{ \infty, \infty, \infty, 4, \infty, \infty \}\} = \min\{\infty,4\} =$ **4**

k	$dist^k[1..7]$						
	1	2	3	4	5	6	7
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

Final Single source shortest path



	$dist^k[1..7]$						
k	1	2	3	4	5	6	7
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3



Optimal Binary Search Tree (OBST)

Optimal Binary Search Tree (OBST)

- A binary search tree is one of the most important data structures in computer science
- Principal applications is to implement a dictionary, a set of elements of a set are with the operations of searching, insertion, and deletion
- If probabilities of searching for elements of a set are known, it is natural to pose a question about an OBST for which the average number of comparisons in a search is the smallest possible

- As an example, consider four keys A, B, C, and D to be searched for with probabilities 0.1, 0.2, 0.4, and 0.3, respectively.
- Figure 8.8 depicts two out of 14 possible binary search trees containing these keys.
- The average number of comparisons in a successful search in the first of these trees is 0.1X1 + 0.2X2 + 0.4X3 + 0.3X4 = 2.9, while for the second one it is 0.1X2 + 0.2X1 + 0.4X2 + 0.3X3 = 2.1.
- Neither of these two trees is, in fact, optimal.
- The total number of binary search trees with *n keys is equal* to the n^{th} *Catalan number.*

$$c(n) = \binom{2n}{n} \frac{1}{n+1} \quad \text{for } n > 0, \quad c(0) = 1,$$

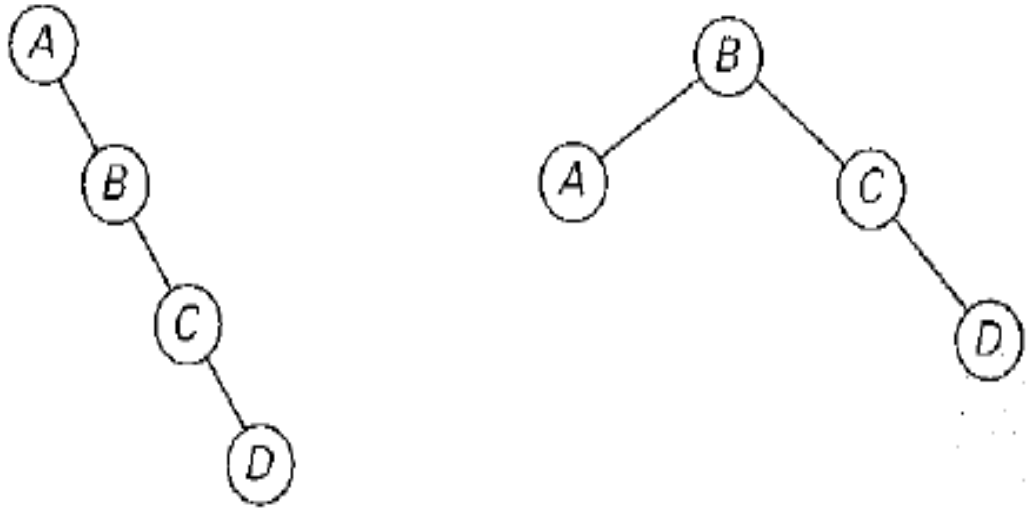
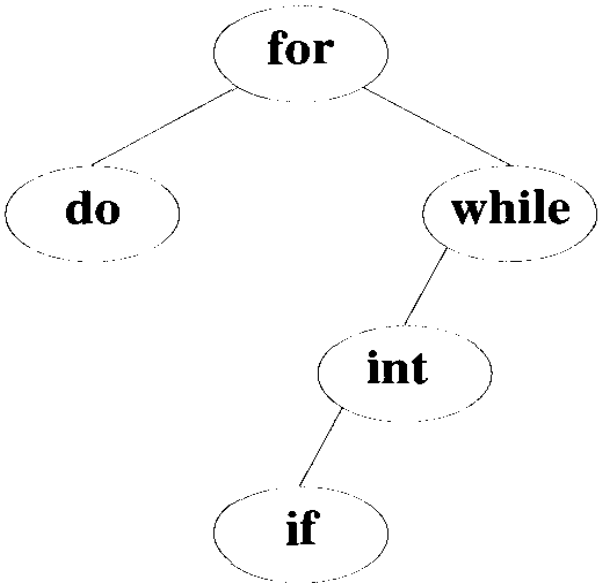
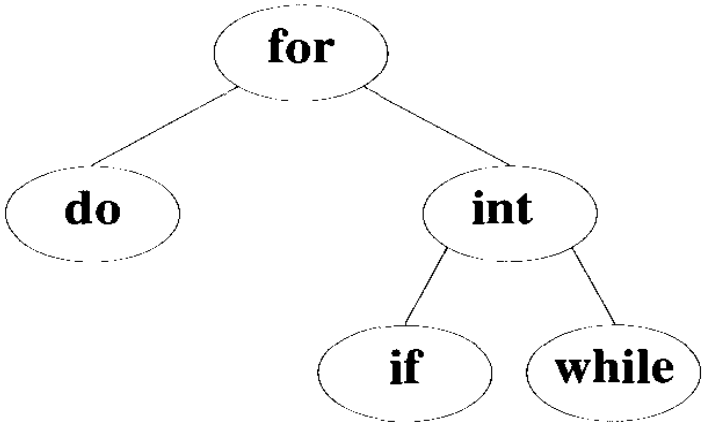


FIGURE 8.8 Two out of 14 possible binary search trees with keys A, B, C, and D



(a)



(b)

In a general situation, we can expect different identifiers to be searched for with different frequencies (or probabilities). In addition, we can expect unsuccessful searches also to be made. Let us assume that the given set of identifiers is $\{a_1, a_2, \ldots, a_n\}$ with $a_1 < a_2 < \cdots < a_n$. Let $p(i)$ be the probability with which we search for a_i. Let $q(i)$ be the probability that the identifier x being searched for is such that $a_i < x < a_{i+1}$, $0 \leq i \leq n$ (assume $a_0 = -\infty$ and $a_{n+1} = +\infty$). Then, $\sum_{0 \leq i \leq n} q(i)$ is the probability of an unsuccessful search. Clearly, $\sum_{1 \leq i \leq n} p(i) + \sum_{0 \leq i \leq n} q(i) = 1$. Given this data, we wish to construct an optimal binary search tree for $\{a_1, a_2, \ldots, a_n\}$. First, of course, we must be precise about what we mean by an optimal binary search tree.

In obtaining a cost function for binary search trees, it is useful to add a fictitious node in place of every empty subtree in the search tree. Such nodes, called external nodes, are drawn square in Figure 5.13. All other nodes are internal nodes. If a binary search tree represents n identifiers, then there will be exactly n internal nodes and $n+1$ (fictitious) external nodes. Every internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.

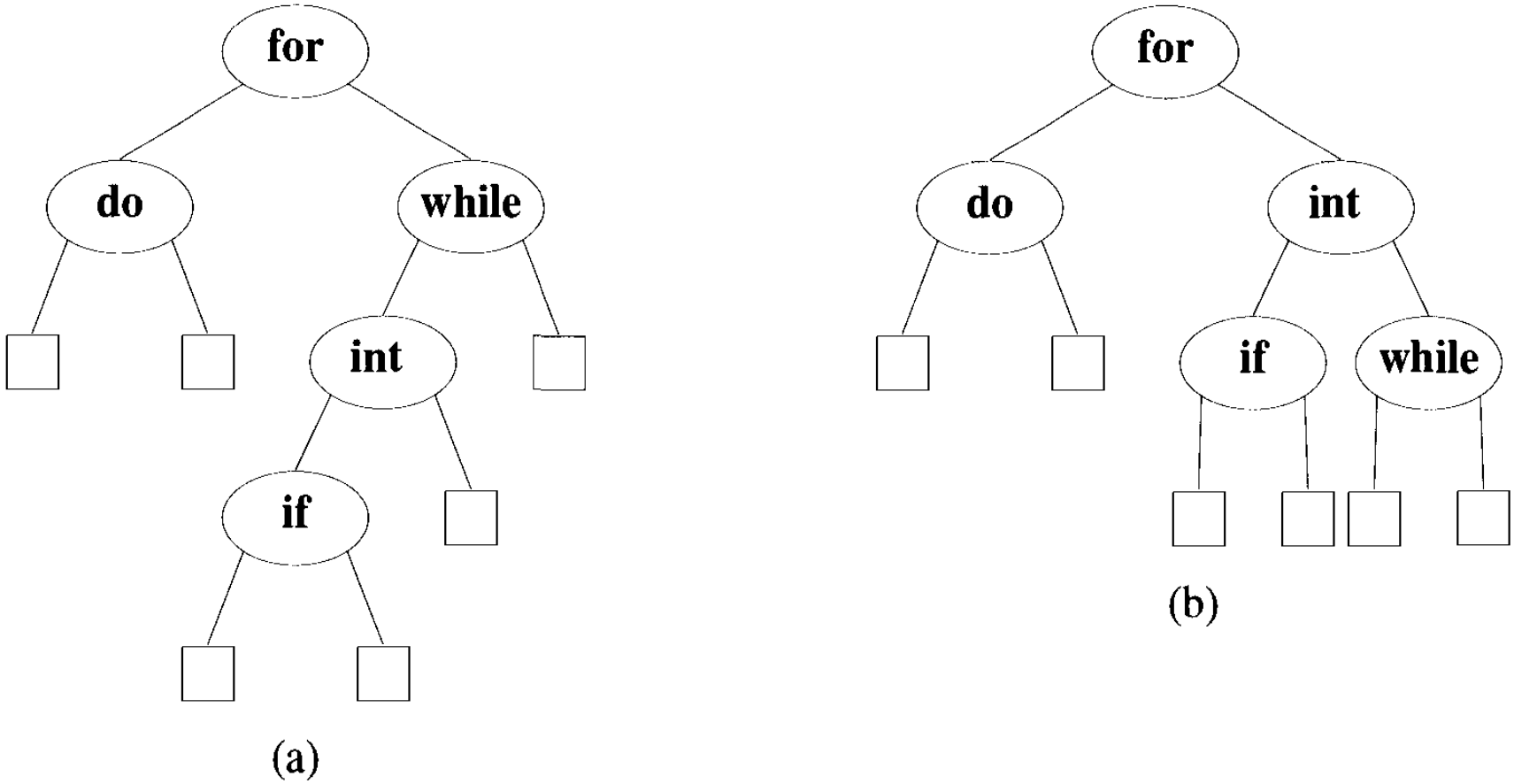


Figure 5.13 Binary search trees of Figure 5.12 with external nodes added

If a successful search terminates at an internal node at level l, then l iterations of the **while** loop of Algorithm 2.5 are needed. Hence, the expected cost contribution from the internal node for a_i is $p(i) * \text{level}(a_i)$.

Unsuccessful searches terminate with $t = 0$ (i.e., at an external node) in algorithm ISearch (Algorithm 2.5). The identifiers not in the binary search tree can be partitioned into $n + 1$ equivalence classes $E_i, 0 \leq i \leq n$. The class E_0 contains all identifiers x such that $x < a_1$. The class E_i contains all identifiers x such that $a_i < x < a_{i+1}$, $1 \leq i < n$. The class E_n contains all identifiers x, $x > a_n$. It is easy to see that for all identifiers in the same class E_i, the search terminates at the same external node. For identifiers in different E_i the search terminates at different external nodes. If the failure

```
1    Algorithm OBST(p, q, n)
2    // Given n distinct identifiers a_1 < a_2 < ··· < a_n and probabilities
3    // p[i], 1 ≤ i ≤ n, and q[i], 0 ≤ i ≤ n, this algorithm computes
4    // the cost c[i, j] of optimal binary search trees t_ij for identifiers
5    // a_{i+1}, ..., a_j. It also computes r[i, j], the root of t_ij.
6    // w[i, j] is the weight of t_ij.
7    {
8         for i := 0 to n − 1 do
9         {
10             // Initialize.
11             w[i, i] := q[i]; r[i, i] := 0; c[i, i] := 0.0;
12             // Optimal trees with one node
13             w[i, i + 1] := q[i] + q[i + 1] + p[i + 1];
14             r[i, i + 1] := i + 1;
15             c[i, i + 1] := q[i] + q[i + 1] + p[i + 1];
16        }
17        w[n, n] := q[n]; r[n, n] := 0; c[n, n] := 0.0;
18        for m := 2 to n do  // Find optimal trees with m nodes.
19             for i := 0 to n − m do
20             {
21                  j := i + m;
22                  w[i, j] := w[i, j − 1] + p[j] + q[j];
23                  // Solve 5.12 using Knuth's result.
24                  k := Find(c, r, i, j);
25                       // A value of l in the range r[i, j − 1] ≤ l
26                       // ≤ r[i + 1, j] that minimizes c[i, l − 1] + c[l, j];
27                  c[i, j] := w[i, j] + c[i, k − 1] + c[k, j];
28                  r[i, j] := k;
29             }
30        write (c[0, n], w[0, n], r[0, n]);
31   }
```

```
1    Algorithm Find(c, r, i, j)
2    {
3         min := ∞;
4         for m := r[i, j − 1] to r[i + 1, j] do
5              if (c[i, m − 1] + c[m, j]) < min then
6              {
7                   min := c[i, m − 1] + c[m, j]; l := m;
8              }
9         return l;
10   }
```

To apply dynamic programming to the problem of obtaining an optimal binary search tree, we need to view the construction of such a tree as the result of a sequence of decisions and then observe that the principle of optimality holds when applied to the problem state resulting from a decision. A possible approach to this would be to make a decision as to which of the a_i's should be assigned to the root node of the tree. If we choose a_k, then it is clear that the internal nodes for $a_1, a_2, \ldots, a_{k-1}$ as well as the external nodes for the classes $E_0, E_1, \ldots, E_{k-1}$ will lie in the left subtree l of the root. The remaining nodes will be in the right subtree r. Define

$$cost(l) = \sum_{1 \le i < k} p(i) * \text{level}(a_i) + \sum_{0 \le i < k} q(i) * (\text{level}(E_i) - 1)$$

and

$$cost(r) = \sum_{k<i\leq n} p(i) * \text{level}(a_i) + \sum_{k<i\leq n} q(i) * (\text{level}(E_i) - 1)$$

In both cases the level is measured by regarding the root of the respective subtree to be at level 1.

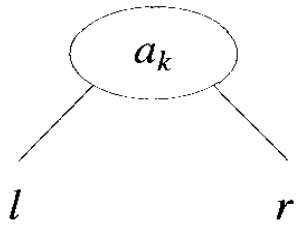


Figure 5.15 An optimal binary search tree with root a_k

Using $w(i, j)$ to represent the sum $q(i) + \sum_{l=i+1}^{j}(q(l) + p(l))$, we obtain the following as the expected cost of the search tree (Figure 5.15):

$$p(k) + cost(l) + cost(r) + w(0, k-1) + w(k, n) \qquad (5.10)$$

If the tree is optimal, then (5.10) must be minimum. Hence, $cost(l)$ must be minimum over all binary search trees containing $a_1, a_2, \ldots, a_{k-1}$ and $E_0, E_1, \ldots, E_{k-1}$. Similarly $cost(r)$ must be minimum. If we use $c(i,j)$ to represent the cost of an optimal binary search tree t_{ij} containing $a_{i+1}, \ldots, a_j$ and $E_i, \ldots, E_j$, then for the tree to be optimal, we must have $cost(l) = c(0, k-1)$ and $cost(r) = c(k,n)$. In addition, k must be chosen such that

$$p(k) + c(0,k-1) + c(k,n) + w(0,k-1) + w(k,n)$$

is minimum. Hence, for $c(0,n)$ we obtain

$$c(0,n) = \min_{1 \le k \le n} \{c(0,k-1) + c(k,n) + p(k) + w(0,k-1) + w(k,n)\} \quad (5.11)$$

We can generalize (5.11) to obtain for any $c(i,j)$

$$c(i,j) = \min_{i < k \le j} \{c(i,k-1) + c(k,j) + p(k) + w(i,k-1) + w(k,j)\}$$

$$c(i,j) = \min_{i<k\leq j}\{c(i,k-1)+c(k,j)\}+w(i,j) \qquad (5.12)$$

Equation 5.12 can be solved for $c(0,n)$ by first computing all $c(i,j)$ such that $j-i=1$ (note $c(i,i)=0$ and $w(i,i)=q(i)$, $0\leq i\leq n$). Next we can compute all $c(i,j)$ such that $j-i=2$, then all $c(i,j)$ with $j-i=3$, and so on. If during this computation we record the root $r(i,j)$ of each tree t_{ij}, then an optimal binary search tree can be constructed from these $r(i,j)$. Note that $r(i,j)$ is the value of k that minimizes (5.12).

Example 5.18 Let $n = 4$ and $(a_1, a_2, a_3, a_4) = (\textbf{do, if, int, while})$. Let $p(1:4) = (3, 3, 1, 1)$ and $q(0:4) = (2, 3, 1, 1, 1)$. The p's and q's have been multiplied by 16 for convenience. Initially, we have $w(i, i) = q(i), c(i, i) = 0$ and $r(i, i) = 0, 0 \leq i \leq 4$. Using Equation 5.12 and the observation $w(i, j) = p(j) + q(j) + w(i, j-1)$, we get

$$
\begin{aligned}
w(0,1) &= p(1) + q(1) + w(0,0) = 8 \\
c(0,1) &= w(0,1) + \min\{c(0,0) + c(1,1)\} = 8 \\
r(0,1) &= 1 \\
w(1,2) &= p(2) + q(2) + w(1,1) = 7 \\
c(1,2) &= w(1,2) + \min\{c(1,1) + c(2,2)\} = 7 \\
r(0,2) &= 2 \\
w(2,3) &= p(3) + q(3) + w(2,2) = 3 \\
c(2,3) &= w(2,3) + \min\{c(2,2) + c(3,3)\} = 3 \\
r(2,3) &= 3 \\
w(3,4) &= p(4) + q(4) + w(3,3) = 3 \\
c(3,4) &= w(3,4) + \min\{c(3,3) + c(4,4)\} = 3 \\
r(3,4) &= 4
\end{aligned}
$$

w(0,2) = p(2)+q(2)+w(0,1) = 3+1+8 = **12**

c(0,2) = w(0,2) + min {c(0,0)+c(1,2), c(0,1)+c(2,2)} = 12 + min{0+7,8+0}

= 12+min{7,8} = 12+7 = **19**

r(0,2) = **1**

w(1,3) = p(3)+q(3)+w(1,2) = 1+1+7 = **9**

c(1,3) = w(1,3) + min {c(1,1)+c(2,3), c(1,2)+c(3,3)} = 9 + min{0+3, 7+0}

= 9+min{3,7} = 9+3 = **12**

r(1,3) = **2**

w(2,4) = p(4)+q(4)+w(2,3) = 1+1+3 = **5**

c(2,4) = w(2,4) + min {c(2,2)+c(3,4), c(2,3)+c(4,4)} = 5 + min{0+3, 3+0}

= 5+min{3,3} = 5+3 = **8**

r(2,4) = **3**

w(0,3) = p(3)+q(3)+w(0,2) = 1+1+12 = **14**

c(0,3) = w(0,3) + min {c(0,0)+c(1,3), c(0,1)+c(2,3), c(0,2)+c(3,3)}

= 14 + min{0+12,8+3, 19+0} = 14 + min{12,11,19} = 14+11 = **25**

r(0,2) = **2**

w(1,4) = p(4)+q(4)+w(1,3) = 1+1+9 = **11**

c(1,4) = w(1,4) + min {c(1,1)+c(2,4), c(1,2)+c(3,4),c(1,3)+c(4,4)}

= 11 + min{0+8, 7+3, 12+0} = 11 + min{8,10,12} = 11 + 8 = **19**

r(1,4) = **2**

w(0,4) = p(4)+q(4)+w(0,3) = 1+1+14 = **16**

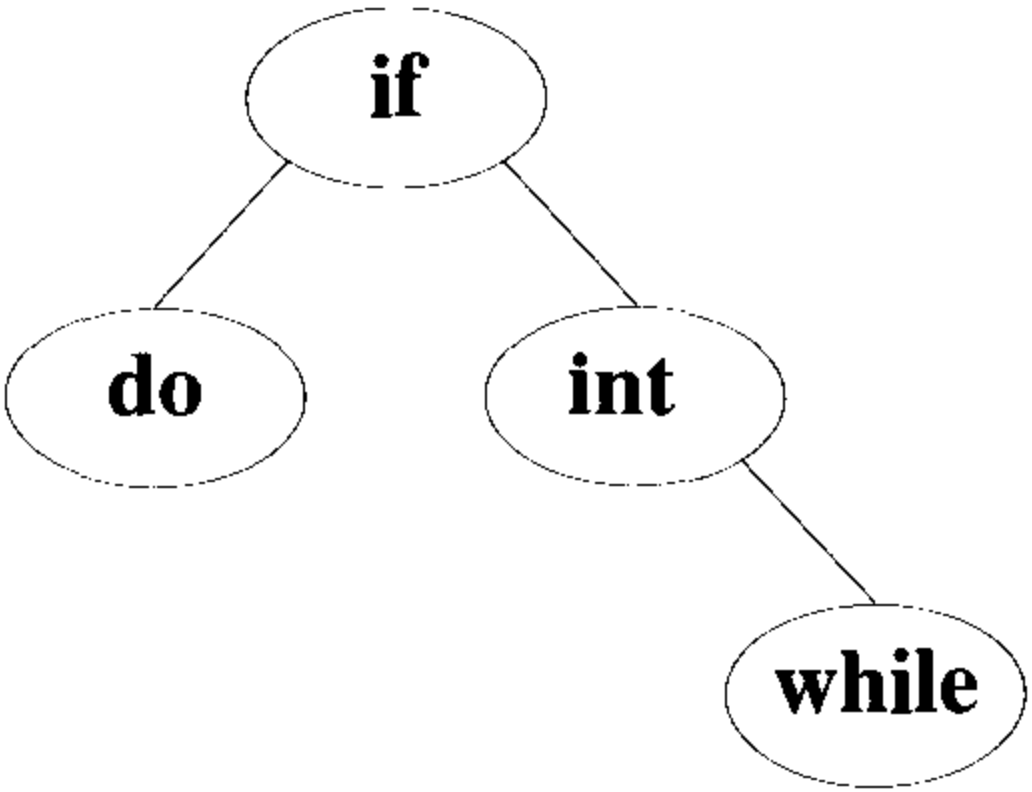
c(0,4) = w(0,4) + min {c(0,0)+c(1,4), c(0,1)+c(2,4), c(0,2)+c(3,4), c(0,3)+c(4,4)}

= 16 + min{0+19,8+8, 19+3, 25+0} = 16 + min{19,16,22,25} = 16+16 = **32**

r(0,4) = **2**

Knowing $w(i, i+1)$ and $c(i, i+1)$, $0 \leq i < 4$, we can again use Equation 5.12 to compute $w(i, i+2)$, $c(i, i+2)$, and $r(i, i+2)$, $0 \leq i < 3$. This process can be repeated until $w(0, 4)$, $c(0, 4)$, and $r(0, 4)$ are obtained. The table of Figure 5.16 shows the results of this computation. The box in row i and column j shows the values of $w(j, j+i)$, $c(j, j+i)$ and $r(j, j+i)$ respectively. The computation is carried out by row from row 0 to row 4. From the table we see that $c(0, 4) = 32$ is the minimum cost of a binary search tree for (a_1, a_2, a_3, a_4). The root of tree t_{04} is a_2. Hence, the left subtree is t_{01} and the right subtree t_{24}. Tree t_{01} has root a_1 and subtrees t_{00} and t_{11}. Tree t_{24} has root a_3; its left subtree is t_{22} and its right subtree t_{34}. Thus, with the data in the table it is possible to reconstruct t_{04}. Figure 5.17 shows t_{04}. □

	0	1	2	3	4
0	$w_{00} = 2$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
1	$w_{01} = 8$ $c_{01} = 8$ $r_{01} = 1$	$w_{12} = 7$ $c_{12} = 7$ $r_{12} = 2$	$w_{23} = 3$ $c_{23} = 3$ $r_{23} = 3$	$w_{34} = 3$ $c_{34} = 3$ $r_{34} = 4$	
2	$w_{02} = 12$ $c_{02} = 19$ $r_{02} = 1$	$w_{13} = 9$ $c_{13} = 12$ $r_{13} = 2$	$w_{24} = 5$ $c_{24} = 8$ $r_{24} = 3$		
3	$w_{03} = 14$ $c_{03} = 25$ $r_{03} = 2$	$w_{14} = 11$ $c_{14} = 19$ $r_{14} = 2$			
4	$w_{04} = 16$ $c_{04} = 32$ $r_{04} = 2$				


if
do
int
while

- Construct the Optimal Binary Search Tree for the following data.

Key	A	B	C	D
Probability	0.1	0.2	0.4	0.3

20 OPTIMAL BINARY SEARCH TREE

Example:

Key	A	B	C	D
Probability	0.1	0.2	0.4	0.3

Recurrance Formula as —

$$C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} P_s \quad \text{for } 1 \le i \le j \le n.$$

Main Table

i ↓ / j →	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

Root Table

i	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					

① → $C[1,2]$ = i=1 j=2 k=2

k=1 & k=2

$$C[1,2] = \min \begin{cases} k=1 \\ C[1,0] + C[2,2] + P_A + P_B = 0 + 0.2 + 0.1 + 0.2 = 0.5 \\ \quad = 0.5 \\ \boxed{k=2} \\ C[1,1] + C[3,2] + P_A + P_B = 0.1 + 0 + 0.1 + 0.2 = \\ \quad = \boxed{0.4} \end{cases}$$

$C[1,2] = \min(0.5, 0.4) = 0.4$ Root Table key = 2.

② → C[2,3] = i=2 j=3 k=2

k=2 & k=3

$$C[2,3] = \min \begin{cases} k=2 \\ C[2,1] + C[3,3] + P_B + P_C = 0 + 0.4 + 0.4 + 0.2 \\ \qquad = 1.0 \\ \boxed{k=3} \\ C[2,2] + C[4,3] + P_B + P_C = 0.2 + 0 + 0.2 + 0.4 \\ \qquad = \boxed{0.8} \end{cases}$$

C B

$C[2,3] = \min(1.0, 0.8) = 0.8$ Root Table key = 3

③ → C[3,4] = i=3 j=4 k=2

k=3 & k=4

$$C[3,4] = \begin{cases} \boxed{k=3} \\ C[3,2] + C[4,4] + P_C + P_D = 0 + 0.3 + 0.4 + 0.3 \\ \qquad = \boxed{1.0} \\ k=4 \\ C[3,3] + C[5,4] + P_C + P_D = 0.4 + 0 + 0.4 + 0.3 \\ \qquad = 1.1 \end{cases}$$

$C[3,4] = \min(1.0, 1.1) = 1.0$ Root Table key = 3

④ → C[1,3] = i=1 j=3 k=3

k=1 k=2 k=3

$$C[1,3] = \min \begin{cases} k=1 \\ C[1,0] + C[2,3] + P_A + P_B + P_C = 0 + 0.8 + 0.1 + 0.2 + 0.4 \\ \qquad = 1.5 \\ k=2 \\ C[1,1] + C[3,3] + P_A + P_B + P_C = 0.1 + 0.4 + 0.1 + 0.2 + 0.4 \\ \qquad = 1.2 \\ \boxed{k=3} \\ C[1,2] + C[4,3] + P_A + P_B + P_C = 0.4 + 0 + 0.1 + 0.2 + 0.4 = \boxed{1.1} \end{cases}$$

$C[1,3] = \min(1.5, 1.2, 1.1) = 1.1$ Root Table key = 3.

⑤ → $C[2,4] =$ $i=2$ $j=4$ $k=3$

$k=2$ $k=3$ $k=4$

$$C[2,4] = \min \begin{cases} k=2 \\ C[2,1] + C[3,4] + P_B + P_C + P_D = 0 + 1.0 + 0.2 + 0.4 + 0.3 \\ \qquad = 1.9 \\ \boxed{k=3} \\ C[2,2] + C[4,4] + P_B + P_C + P_D = 0.2 + 0.3 + 0.2 + 0.4 + 0.3 \\ k=4 \qquad = \boxed{1.4} \\ C[2,3] + C[5,4] + P_B + P_C + P_D = 0.8 + 0 + 0.2 + 0.4 + 0.3 \\ \qquad = 1.7 \end{cases}$$

$C[2,4] = \min(1.9, 1.4, 1.7) = 1.4$ Root Table key = 3.

⑥ → $C[1,4] =$ $i=1$ $j=4$ $k=4$

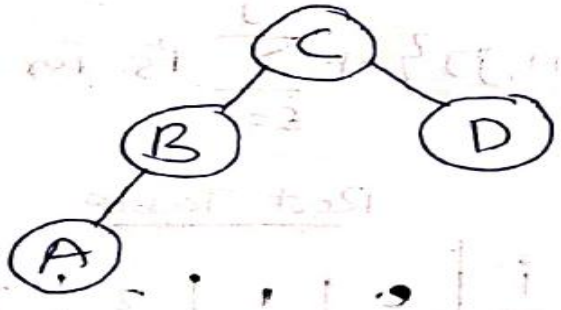
$k=1$ $k=2$ $k=3$ $k=4$

$$C[1,4] = \min \begin{cases} k=1 \\ C[1,0] + C[2,4] + P_A + P_B + P_C + P_D = 0 + 1.4 + 0.1 + 0.2 + 0.4 + 0.3 \\ \qquad = 2.4 \\ k=2 \\ C[1,1] + C[3,4] + P_A + P_B + P_C + P_D = 0.1 + 1.0 + 0.1 + 0.2 + 0.4 + 0.3 \\ \qquad = 2.1 \\ \boxed{k=3} \\ C[1,2] + C[4,4] + P_A + P_B + P_C + P_D = 0.4 + 0.3 + 0.1 + 0.2 + 0.3 \\ \qquad = \boxed{1.3} \\ k=4 \\ C[1,3] + C[5,4] + P_A + P_B + P_C + P_D = 1.1 + 0 + 0.1 + 0.2 + 0.3 \\ \qquad = 2.1 \end{cases}$$

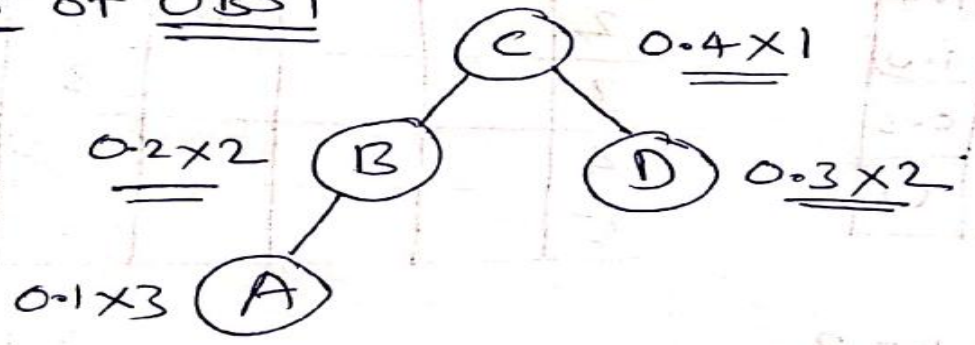
$C[1,4] = \min(2.4, 2.1, 1.7, 2.1) = 1.7$ Root Table key = 3

Optimal BST for Given Example –

Root Element = 3 = C



Proof of OBST



$= 0.4 + 0.4 + 0.6 + 0.3$

$= 1.7$

Travelling Sales Person Problem

Travelling Sales Person Problem

- Let G = (V, E) be a directed graph with edge costs c_{ij}.
- The variable c_{ij} is defined such that $c_{ij} > 0$ for all i and j and $c_{ij} = \infty$ if $\langle i,j \rangle \notin E$.
- Let $|V| = n$ and assume $n > 1$.
- A tour of G is a directed simple cycle that includes every vertex in V.
- The cost of a tour is the sum of the cost of the edges on the tour.
- The travelling sales person problem is to find a tour of minimum cost.
- Example:- A postal van to pick up mail from mail boxes located at n different sites where one vertex represent the post office from which the postal van starts and to which it must return

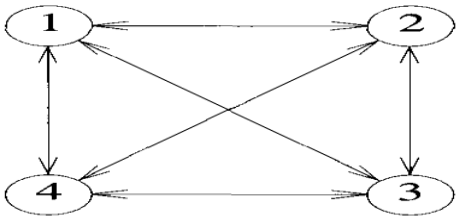
In the following discussion we shall, without loss of generality, regard a tour to be a simple path that starts and ends at vertex 1. Every tour consists of an edge $\langle 1, k \rangle$ for some $k \in V - \{1\}$ and a path from vertex k to vertex 1. The path from vertex k to vertex 1 goes through each vertex in $V - \{1, k\}$ exactly once. It is easy to see that if the tour is optimal, then the path from k to 1 must be a shortest k to 1 path going through all vertices in $V - \{1, k\}$. Hence, the principle of optimality holds. Let $g(i, S)$ be the length of a shortest path starting at vertex i, going through all vertices in S, and terminating at vertex 1. The function $g(1, V - \{1\})$ is the length of an optimal salesperson tour. From the principal of optimality it follows that

$$g(1, V - \{1\}) = \min_{2 \leq k \leq n} \{c_{1k} + g(k, V - \{1, k\})\} \tag{5.20}$$

Generalizing (5.20), we obtain (for $i \notin S$)

$$g(i, S) = \min_{j \in S} \{c_{ij} + g(j, S - \{j\})\} \tag{5.21}$$

Equation 5.20 can be solved for $g(1, V - \{1\})$ if we know $g(k, V - \{1, k\})$ for all choices of k. The g values can be obtained by using (5.21). Clearly, $g(i, \phi) = c_{i1}$, $1 \leq i \leq n$. Hence, we can use (5.21) to obtain $g(i, S)$ for all S of size 1. Then we can obtain $g(i, S)$ for S with $|S| = 2$, and so on. When $|S| < n - 1$, the values of i and S for which $g(i, S)$ is needed are such that $i \neq 1$, $1 \notin S$, and $i \notin S$.



(a)

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

(b)

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\})\}$$

When set |S|=1 then

$g(2,\phi) = c_{21} = 5 \quad g(3,\phi) = c_{31} = 6 \quad g(4,\phi) = c_{41} = 8$

When set |S|=2 then

$g(2,\{3\}) = c_{23} + g(3,\phi) = 9 + 6 = 15$

$g(2,\{4\}) = c_{24} + g(4,\phi) = 10 + 8 = 18$

$g(3,\{2\}) = c_{32} + g(2,\phi) = 13 + 5 = 18$

$g(3,\{4\}) = c_{34} + g(4,\phi) = 12 + 8 = 20$

$g(4,\{2\}) = c_{42} + g(2,\phi) = 8 + 5 = 13$

$g(4,\{3\}) = c_{43} + g(3,\phi) = 9 + 6 = 15$

When set |S|=3 then

$g(2, \{3, 4\}) = \min \{ c_{23} + g(3, \{4\}), c_{24} + g(4, \{3\}) \}$

$= \min \{ 9+20, 10+15\}$

$= \min \{ 29, 25\} = \mathbf{25}$

$g(3, \{2, 4\}) = \min \{ c_{32} + g(2, \{4\}), c_{34} + g(4, \{2\}) \}$

$= \min \{ 13+18, 12+13\}$

$= \min \{ 31, 25\} = \mathbf{25}$

$g(4, \{2, 3\}) = \min \{ c_{42} + g(2, \{3\}), c_{43} + g(3, \{2\}) \}$

$= \min \{ 8+15, 9+18\}$

$= \min \{ 23, 27\} = \mathbf{23}$

Finally when |S|=4 then

$g(1,\{2,3,4\}) = \min\{ c_{12}+g(2, \{3,4\}),$

$c_{13}+g(3, \{2,4\}),$

$c_{14}+g(4, \{2,3\}) \}$

$= \min \{ 10+25, 15+25, 20+23 \}$

$= \min \{ 35, 40, 43\}$

$= \mathbf{35}$

Optimal Path from source vertex 1 as

J(1, {2,3,4}) = 2 tour starts from 1 and goes to 2

J(2, {3,4}) = 4 then from 2 to 4

J(4,{3}) = 3 then from 4 to 3

Hence, the optimal tour is

1, 2, 4, 3, 1

- Solve the following Travelling Salesperson problem represented as a graph shown in figure using Dynamic Programming.

When the set |S| = 1,

$g(2, \phi) = c_{21} = 30$

$g(3, \phi) = c_{31} = 4$

$g(4, \phi) = c_{41} = 6$

When the set |S| = 2,

$g(2, \{3\}) = c_{23} + g(3, \phi) = 10+4=14$

$g(2, \{4\}) = c_{24} + g(4, \phi) = 5+6=11$

$g(3, \{2\}) = c_{32} + g(2, \phi) = 10+30=40$

$g(3, \{4\}) = c_{34} + g(4, \phi) = 20+6=26$

$g(4, \{2\}) = c_{42} + g(2, \phi) = 5+30=35$

$g(4, \{3\}) = c_{43} + g(3, \phi) = 20+4=24$

When the set |S| = 3,

g(2, {3,4}) = min { c_{23} + g(3, {4}), c_{24} + g(4, {3}) = min{ 10+26, 5+24}= min {36,29} = 29

g(3, {2,4}) = min { c_{32} + g(2, {4}), c_{34} + g(4, {2}) = min{ 10+11, 20+35}= min {21,55} = 21

g(4, {2,3}) = min { c_{42} + g(2, {3}), c_{43} + g(3, {2}) = min{ 5+14, 20+40}= min {19,60} = 19

When the set |S| = 4,

g(1, {2,3,4}) = min { c_{12} + g(2, {3,4}), c_{13} + g(3, {2,4}, c_{14} + g(4, {2,3})

= min{ 30+29, 4+21, 6+19 }= min {59,25, 25} = 25

Optimal Path from source vertex 1 as

J(1, {2,3,4}) = 3 tour starts from 1 and goes to 3

J(3, {2,4}) = 2 then from 3 to 2

J(2,{4}) = 4 then from 2 to 4

Hence, the optimal tour is

1, 3, 2, 4, 1